VAX/VMS Run-Time Library Routines Reference Manual (continued)

Order No. AA–Z505C–TE (Volume 8C)

QLZ55–GZ (Volumes 8B and 8C)

digital

software

Part II Run-Time Library Routines (continued)

This part contains descriptions of the Run-Time Library routines.

Order Number: AA-Z505C-TE (Volume 5D)
Order Number: QLZ55–GZ (Volumes 5C and 5D)

MTH$xACOS—Arc Cosine of Angle Expressed in Radians

Given the cosine of an angle, MTH$xACOS returns that angle (in radians).

FORMAT

MTH$ACOS *x*
MTH$DACOS *x*
MTH$GACOS *x*
MTH$HACOS *h_radians ,x*

Each of the above four formats accepts as input one of the four floating-point types.

jsb entries

MTH$ACOS_R4
MTH$DACOS_R7
MTH$GACOS_R7
MTH$HACOS_R8

Each of the above four JSB entries accepts as input one of the four floating-point types.

RETURNS

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating**
access: **write only**
mechanism: **by value**

Angle in radians. The angle returned will be a value between 0 and PI. MTH$ACOS returns an F_floating number. MTH$DACOS returns a D_floating number. MTH$GACOS returns a G_floating number. Unlike the other three routines, MTH$HACOS returns the angle by reference in the **h_radians** argument.

ARGUMENTS

x

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating, H_floating**
access: **read only**
mechanism: **by reference**

The cosine of the angle whose value (in radians) is to be returned. The **x** argument is the address of a floating-point number that is this cosine. The absolute value of **x** must be less than or equal to 1. For MTH$ACOS, **x** specifies an F_floating number. For MTH$DACOS, **x** specifies a D_floating number. For MTH$GACOS, **x** specifies a G_floating number. For MTH$HACOS, **x** specifies an H_floating number.

h_radians

VMS Usage: **floating_point**
type: **H_floating**
access: **write only**
mechanism: **by reference**

Angle (in radians) whose cosine is specified by **x**. The **h_radians** argument is the address of an H_floating number that is this angle. MTH$HACOS writes the address of the angle into **h_radians**. The **h_radians** argument is used only by the MTH$HACOS routine.

DESCRIPTION

The angle in radians whose cosine is **x** is computed as:

Value of x	Value Returned
0	*PI/2*
1	*0*
-1	*PI*
$0 < X < 1$	*zATAN(zSQRT(1-X²)/X)*, where *zATAN* and *zSQRT* are the Math Library arc tangent and square root routines, respectively, of the appropriate data type
$-1 < X < 0$	*zATAN(zSQRT(1-X²)/X) + PI*
$1 < \|X\|$	The error MTH$_INVARGMAT is signaled

CONDITION VALUES SIGNALED

SS$_ROPRAND	Reserved operand. The MTH$xACOS routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of one and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.
MTH$_INVARGMAT	Invalid argument. The absolute value of **x** is greater than 1. LIB$SIGNAL copies the floating-point reserved operand to the mechanism argument vector CHF$L_MCH_SAVR0/R1. The result is the floating-point reserved operand unless you have written a condition handler to change CHF$L_MCH_SAVR0/R1.

EXAMPLES

1

```
100     !+
        ! This BASIC program demonstrates the use of
        ! MTH$ACOS.
        !-
        EXTERNAL REAL FUNCTION MTH$ACOS
        DECLARE REAL COS_VALUE, ANGLE
300     INPUT "Cosine value between -1 and +1 "; COS_VALUE
400     IF (COS_VALUE < -1) OR (COS_VALUE > 1)
                THEN PRINT "Invalid cosine value"
                        GOTO 300
500     ANGLE = MTH$ACOS( COS_VALUE )
        PRINT "The angle with that cosine is "; ANGLE; "radians"
32767   END
```

This BASIC program prompts for a cosine value and determines the angle that has that cosine. The output generated by this program is as follows:

```
$ RUN ACOS
Cosine value betwen -1 and +1 ?  .5
The angle with that cosine is 1.0472 radians
```

2

```
PROGRAM GETANGLE(INPUT,OUTPUT);
{+}
{ This PASCAL program uses MTH$ACOS to determine
{ the angle which has the cosine given as input.
{-}
VAR
        COS : REAL;
FUNCTION MTH$ACOS(COS : REAL) : REAL;
        EXTERN;
BEGIN
        WRITE('Cosine value between -1 and +1:  ');
        READ (COS);
        WRITELN('The angle with that cosine is ', MTH$ACOS(COS),
        ' radians');
END.
```

This PASCAL program prompts for a cosine value and determines the angle that has that cosine. The output generated by this program is as follows:

```
$ RUN ACOS
Cosine value between -1 and +1:  .5
The angle with that cosine is 1.04720E+00 radians
```

MTH$xACOSD—Arc Cosine of Angle Expressed in Degrees

Given the cosine of an angle, MTH$xACOSD returns that angle (in degrees).

FORMAT

MTH$ACOSD *x*
MTH$DACOSD *x*
MTH$GACOSD *x*
MTH$HACOSD *h_degrees ,x*

Each of the above four formats accepts as input one of the four floating-point types.

jsb entries

MTH$ACOSD_R4
MTH$DACOSD_R7
MTH$GACOSD_R7
MTH$HACOSD_R8

Each of the above JSB entries accepts as input one of the four floating-point types.

RETURNS

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating**
access: **write only**
mechanism: **by value**

Angle in degrees. The angle returned will be a value between 0 and 180. MTH$ACOSD returns an F_floating number. MTH$DACOSD returns a D_floating number. MTH$GACOSD returns a G_floating number. Unlike the other three routines, MTH$HACOSD returns the angle by reference in the **h_degrees** argument.

ARGUMENTS

x

VMS Usage: **floating_point**
type: **F_floating, G_floating, D_floating, H_floating**
access: **read only**
mechanism: **by reference**
Cosine of the angle whose value (in degrees) is to be returned. The **x** argument is the address of a floating-point number that is this cosine. The absolute value of **x** must be less than or equal to 1. For MTH$ACOSD, **x** specifies an F_floating number. For MTH$DACOSD, **x** specifies a D_floating number. For MTH$GACOSD, **x** specifies a G_floating number. For MTH$HACOSD, **x** specifies an H_floating number.

h_degrees

VMS Usage: **floating_point**
type: **H_floating**
access: **write only**
mechanism: **by reference**

Angle (in degrees) whose cosine is specified by **x**. The h_degrees argument is the address of an H_floating number that is this angle. MTH$HACOSD writes the address of the angle into **h_degrees**. The **h_degrees** argument is used only by the MTH$HACOSD routine.

DESCRIPTION

The angle in degrees whose cosine is **X** is computed as:

Value of x	Angle returned
0	*90*
1	*0*
-1	*180*
$0 < X < 1$	*zATAND(zSQRT(1-X²)/X)* , where *zATAND* and *zSQRT* are the Math Library arc tangent and square root routines, respectively, of the appropriate data type
$-1 < X < 0$	*zATAND(zSQRT(1-X²)/X) + 180*
$1 < \|X\|$	The error MTH$_INVARGMAT is signaled

CONDITION VALUES SIGNALED

SS$_ROPRAND	Reserved operand. The MTH$xACOSD routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of one and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.
MTH$_INVARGMAT	Invalid argument: The absolute value of **x** is greater than 1. LIB$SIGNAL copies the floating-point reserved operand to the mechanism argument vector CHF$L_MCH_SAVR0/R1. The result is the floating-point reserved operand unless you have written a condition handler to change CHF$L_MCH_SAVR0/R1.

EXAMPLE

```
PROGRAM ACOSD(INPUT,OUTPUT);
{+}
{ This PASCAL program demonstrates the use of
{ MTH$ACOSD.
{-}
FUNCTION MTH$ACOSD(COS : REAL): REAL; EXTERN;
VAR
  COSINE : REAL;
  RET_STATUS : REAL;
BEGIN
  COSINE := 0.5;
  RET_STATUS := MTH$ACOSD(COSINE);
  WRITELN('The angle, in degress, is: ', RET_STATUS);
END.
```

The output generated by this PASCAL example program is as follows:

```
The angle, expressed in degrees, is:  6.00000E+01
```

MTH$xASIN—Arc Sine in Radians

Given the sine of an angle, MTH$xASIN returns that angle (in radians).

FORMAT

MTH$ASIN *x*
MTH$DASIN *x*
MTH$GASIN *x*
MTH$HASIN *h_radians ,x*

Each of the above four formats accepts as input one of the four floating-point types.

jsb entries

MTH$ASIN_R4
MTH$DASIN_R7
MTH$GASIN_R7
MTH$HASIN_R8

Each of the above four JSB entries accepts as input one of the four floating-point types.

RETURNS

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating,**
access: **write only**
mechanism: **by value**

Angle in radians. The angle returned will be a value between -PI/2 and +PI/2. MTH$ASIN returns an F_floating number. MTH$DASIN returns a D_floating number. MTH$GASIN returns a G_floating number. Unlike the other three routines, MTH$HASIN returns the angle by reference in the **h_radians** argument.

ARGUMENTS

x

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating, H_floating**
access: **read only**
mechanism: **by reference**

The sine of the angle whose value (in radians) is to be returned. The **x** argument is the address of a floating-point number that is this sine. The absolute value of **x** must be less than or equal to 1. For MTH$ASIN, **x** specifies an F_floating number. For MTH$DASIN, **x** specifies a D_floating number. For MTH$GASIN, **x** specifies a G_floating number. For MTH$HASIN, **x** specifies an H_floating number.

h_radians

VMS Usage: **floating_point**
type: **H_floating**
access: **write only**
mechanism: **by reference**

Angle (in radians) whose sine is specified by **x**. The **h_radians** argument is the address of an H_floating number that is this angle. MTH$HASIN writes the address of the angle into **h_radians**. The **h_radians** argument is used only by the MTH$HASIN routine.

DESCRIPTION

The angle in radians whose sine is **x** is computed as:

Value of x	Angle returned
0	*0*
1	*PI/2*
-1	*-PI/2*
$0 < \lvert X \rvert < 1$	$zATAN(X/zSQRT(1-X^2))$, where *zATAN* and *zSQRT* are the Math Library arc tangent and square root routines, respectively, of the appropriate data type
$1 < \lvert X \rvert$	The error MTH$_INVARGMAT is signaled

CONDITION VALUES SIGNALED

SS$_ROPRAND	Reserved operand. The MTH$xASIN routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.
MTH$_INVARGMAT	Invalid argument: The absolute value of **x** is greater than 1. LIB$SIGNAL copies the floating-point reserved operand to the mechanism argument vector CHF$L_MCH_SAVR0/R1. The result is the floating-point reserved operand unless you have written a condition handler to change CHF$L_MCH_SAVR0/R1.

MTH$xASIND—Arc Sine in Degrees

Given the sine of an angle, MTH$xASIND returns that angle (in degrees).

FORMAT

MTH$ASIND *x*
MTH$DASIND *x*
MTH$GASIND *x*
MTH$HASIND *h_degrees ,x*

Each of the above four formats accepts as input one of the four floating-point types.

jsb entries

MTH$ASIND_R4
MTH$DASIND_R7
MTH$GASIND_R7
MTH$HASIND_R8

Each of the above four JSB entries accepts as input one of the four floating-point types.

RETURNS

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating, H_floating**
access: **write only**
mechanism: **by value**

Angle in degrees. The angle returned will be a value between -90 and +90. MTH$ASIND returns an F_floating number. MTH$DASIND returns a D_floating number. MTH$GASIND returns a G_floating number. Unlike the other three routines, MTH$HASIND returns the angle by reference in the **h_degrees** argument.

ARGUMENTS

x

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating, H_floating**
access: **read only**
mechanism: **by reference**

Sine of the angle whose value (in degrees) is to be returned. The **x** argument is the address of a floating-point number that is this sine. The absolute value of **x** must be less than or equal to 1. For MTH$ASIND, **x** specifies an F_floating number. For MTH$DASIND, **x** specifies a D_floating number. For MTH$GASIND, **x** specifies a G_floating number. For MTH$HASIND, **x** specifies an H_floating number.

h_degrees

VMS Usage: **floating_point**
type: **H_floating**
access: **write only**
mechanism: **by reference**

Angle (in degrees) whose cosine is specified by **x**. The **h_degrees** argument is the address of an H_floating number that is this angle. MTH$HASIND writes the address of the angle into **h_degrees**. The **h_degrees** argument is used only by the MTH$HASIND routine.

DESCRIPTION

The angle in degrees whose sine is X is computed as:

Value of x	Value returned
0	*0*
1	*90*
-1	*-90*
$0 < \|X\| < 1$	$zATAND(X/zSQRT(1-X^2))$, where *zATAND* and *zSQRT* are the Math Library arc tangent and square root routines, respectively, of the appropriate data type
$1 < \|X\|$	The error MTH$_INVARGMAT is signaled

CONDITION VALUES SIGNALED

SS$_ROPRAND	Reserved operand. The MTH$xASIND routine encountered a floating point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of one and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.
MTH$_INVARGMAT	Invalid argument: The absolute value of x is greater than 1. LIB$SIGNAL copies the floating-point reserved operand to the mechanism argument vector CHF$L_MCH_SAVR0/R1. The result is the floating-point reserved operand unless you have written a condition handler to change CHF$L_MCH_SAVR0/R1.

MTH$xATAN—Arc Tangent in Radians

Given the tangent of an angle, MTH$xATAN returns that angle (in radians).

FORMAT

MTH$ATAN *x*
MTH$DATAN *x*
MTH$GATAN *x*
MTH$HATAN *h_radians ,x*

Each of the above four formats accepts as input one of the four floating-point types.

jsb entries

MTH$ATAN_R4
MTH$DATAN_R7
MTH$GATAN_R7
MTH$HATAN_R8

Each of the above four JSB entries accepts as input one of the four floating-point types.

RETURNS

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating**
access: **write only**
mechanism: **by value**

Angle in radians. The angle returned will be a value between -PI/2 and +PI/2. MTH$ATAN returns an F_floating number. MTH$DATAN returns a D_floating number. MTH$GATAN returns a G_floating number. Unlike the other three routines, MTH$HATAN returns the angle by reference in the **h_radians** argument.

ARGUMENTS

x

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating, H_floating**
access: **read only**
mechanism: **by reference**

The tangent of the angle whose value (in radians) is to be returned. The **x** argument is the address of a floating-point number that is this tangent. For MTH$ATAN, **x** specifies an F_floating number. For MTH$DATAN, **x** specifies a D_floating number. For MTH$GATAN, **x** specifies a G_floating number. For MTH$HATAN, **x** specifies an H_floating number.

h_radians

VMS Usage: **floating_point**
type: **H_floating**
access: **write only**
mechanism: **by reference**

Angle (in radians) whose tangent is specified by x. The **h_radians** argument is the address of an H_floating number that is this angle. MTH$HATAN writes the address of the angle into **h_radians**. The **h_radians** argument is used only by the MTH$HATAN routine.

DESCRIPTION

In radians, the computation of the arc tangent function is based on the following identities:

```
arctan(X) = X - X^3/3 + X^5/5 - X^7/7 + ...
arctan(X) = X + X*Q(X^2),
     where Q(Y) = - Y/3 + Y^2/5 - Y^3/7 + ...
arctan(X) = X*P(X^2),
     where P(Y) = 1 - Y/3 + Y^2/5 - Y^3/7 + ...
arctan(X) = PI/2 - arctan(1/X)
arctan(X) = arctan(A) + arctan((X-A)/(1+A*X)) for any real A
```

The angle in radians whose tangent is X is computed as:

Value of *X*	Angle Returned
$0 =< X =< 3/32$	$X + X * Q(X^2)$
$3/32 < X =< 11$	$ATAN(A) + V * (P(V^2))$, where *A* and *ATAN(A)* are chosen by table lookup and $V = (X - A)/(1 + A*X)$
$11 < X$	$PI/2 - W * (P(W^2))$ where $W = 1/X$
$X < 0$	$-zATAN(\|X\|)$

CONDITION VALUE SIGNALED

SS$_ROPRAND	Reserved operand. The MTH$xATAN routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.

MTH$xATAND—Arc Tangent in Degrees

Given the tangent of an angle, MTH$xATAND returns that angle (in degrees).

FORMAT

MTH$ATAND *x*
MTH$DATAND *x*
MTH$GATAND *x*
MTH$HATAND *h_degrees ,x*

Each of the above four formats accepts as input one of the four floating-point types.

jsb entries

MTH$ATAND_R4
MTH$DATAND_R7
MTH$GATAND_R7
MTH$HATAND_R8

Each of the above four JSB entries accepts as input one of the four floating-point types.

RETURNS

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating**
access: **write only**
mechanism: **by value**

Angle in degrees. The angle returned will be a value between -90 and +90. MTH$ATAND returns an F_floating number. MTH$DATAND returns a D_floating number. MTH$GATAND returns a G_floating number. Unlike the other three routines, MTH$HATAND returns the angle by reference in the **h_degrees** argument.

ARGUMENTS

x

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating, H_floating**
access: **read only**
mechanism: **by reference**

The tangent of the angle whose value (in degrees) is to be returned. The **x** argument is the address of a floating-point number that is this tangent. For MTH$ATAND, **x** specifies an F_floating number. For MTH$DATAND, **x** specifies a D_floating number. For MTH$GATAND, **x** specifies a G_floating number. For MTH$HATAND, **x** specifies an H_floating number.

h_degrees

VMS Usage: **floating_point**
type: **H_floating**
access: **write only**
mechanism: **by reference**

Angle (in degrees) whose tangent is specified by x. The **h_degrees** argument is the address of an H_floating number that is this angle. MTH$HATAND writes the address of the angle into **h_degrees**. The **h_degrees** argument is used only by the MTH$HATAND routine.

DESCRIPTION

The computation of the arc tangent function is based on the following identities:

$$\text{arctand}(X) = 180/PI(X - X^3/3 + X^5/5 - X^7/7 + \ldots)$$

$$\text{arctand}(X) = 64*X + X*Q(X^2),$$
$$\text{where } Q(Y) = 180/PI*[(1- 64*PI/180)] - Y/3 + Y^2/5 - Y^3/7 + Y^4/9 \ldots]$$

$$\text{arctand}(X) = X*P(X^2),$$
$$\text{where } P(Y) = 180/PI*[1 - Y/3 + Y^2/5 - Y^3/7 + Y^4/9 \ldots]$$

$$\text{arctand}(X) = 90 - \text{arctand}(1/X)$$

$$\text{arctand}(X) = \text{arctand}(A) + \text{arctand}((X - A)/(1 + A*X))$$

The angle in degrees whose tangent is X is computed as:

Tangent	Angle Returned
$X =< 3/32$	$64*X + X*Q(X^2)$
$3/32 < X =< 11$	$ATAND(A) + V*P(V^2)$, where A and $ATAND(A)$ are chosen by table lookup and $V = (X - A)/(1 + A*X)$
$11 < X$	$90 - W * (P(W^2))$, where $W = 1/X$
$X < 0$	$-zATAND(\|X\|)$

CONDITION VALUE SIGNALED

SS$_ROPRAND

Reserved operand. The MTH$xATAND routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.

MTH$xATAN2—Arc Tangent in Radians with Two Arguments

Given **y** and **x**, MTH$xATAN2 returns the angle (in radians) whose tangent is given by the quotient of **y** and **x**, (**y**/**x**).

FORMAT

MTH$ATAN2 *y ,x*
MTH$DATAN2 *y ,x*
MTH$GATAN2 *y ,x*
MTH$HATAN2 *h_radians ,y ,x*

Each of the above four formats accepts as input one of the four floating-point types.

RETURNS

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating**
access: **write only**
mechanism: **by value**

Angle in radians. MTH$ATAN2 returns an F_floating number. MTH$DATAN2 returns a D_floating number. MTH$GATAN2 returns a G_floating number. Unlike the other three routines, MTH$HATAN2 returns the angle by reference in the **h_radians** argument.

ARGUMENTS

y

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating, H_floating**
access: **read only**
mechanism: **by reference**
Dividend. The **y** argument is the address of a floating-point number that is this dividend. For MTH$ATAN2, **y** specifies an F_floating number. For MTH$DATAN2, **y** specifies a D_floating number. For MTH$GATAN2, **y** specifies a G_floating number. For MTH$HATAN2, **y** specifies an H_floating number.

x

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating, H_floating**
access: **read only**
mechanism: **by reference**
Divisor. The **x** argument is the address of a floating-point number that is this divisor. For MTH$ATAN2, **x** specifies an F_floating number. For MTH$DATAN2, **x** specifies a D_floating number. For MTH$GATAN2, **x** specifies a G_floating number. For MTH$HATAN2, **x** specifies an H_floating number.

h_radians

VMS Usage: **floating_point**
type: **H_floating**
access: **write only**
mechanism: **by reference**

Angle (in radians) whose tangent is specified by (y/x). The **h_radians** argument is the address of an H_floating number that is this angle. MTH$HATAN2 writes the address of the angle into **h_radians**. The **h_radians** argument is used only by the MTH$HATAN2 routine.

DESCRIPTION

The angle in radians whose tangent is Y/X is computed as follows and *f* is defined in the description of MTH$zCOSH.

Value of Input Arguments	Angle Returned
$X = 0$ or $Y/X > 2^{(f+1)}$	$PI/2 * (sign\ Y)$
$X > 0$ and $Y/X =< 2^{(f+1)}$	$zATAN(Y/X)$
$X < 0$ and $Y/X =< 2^{(f+1)}$	$PI * (sign\ Y) + zATAN(Y/X)$

CONDITION VALUES SIGNALED

SS$_ROPRAND	Reserved operand. The MTH$xATAN2 routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.
MTH$_INVARGMAT	Invalid argument: Both x and y are zero. LIB$SIGNAL copies the floating-point reserved operand to the mechanism argument vector CHF$L_MCH_SAVR0/R1. The result is the floating-point reserved operand unless you have written a condition handler to change CHF$L_MCH_SAVR0/R1.

MTH$xATAND2—Arc Tangent in Degrees with Two Arguments

Given **y** and **x**, MTH$xATAND2 returns the angle (in degrees) whose tangent is given by the quotient of **y** and **x**, **(y/x)**.

FORMAT

MTH$ATAND2 *y ,x*
MTH$DATAND2 *y ,x*
MTH$GATAND2 *y ,x*
MTH$HATAND2 *h_degrees ,y ,x*

Each of the above four formats accepts as input one of the four floating-point types.

RETURNS

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating**
access: **write only**
mechanism: **by value**

Angle (in degrees). MTH$ATAND2 returns an F_floating number. MTH$DATAND2 returns a D_floating number. MTH$GATAND2 returns a G_floating number. Unlike the other three routines, MTH$HATAND2 returns the angle by reference in the **h_degrees** argument.

ARGUMENTS

y

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating, H_floating**
access: **read only**
mechanism: **by reference**

Dividend. The **y** argument is the address of a floating-point number that is this dividend. For MTH$ATAND2, **y** specifies an F_floating number. For MTH$DATAND2, **y** specifies a D_floating number. For MTH$GATAND2, **y** specifies a G_floating number. For MTH$HATAND2, **y** specifies an H_floating number.

x

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating, H_floating**
access: **read only**
mechanism: **by reference**

Divisor. The **x** argument is the address of a floating-point number that is this divisor. For MTH$ATAND2, **x** specifies an F_floating number. For MTH$DATAND2, **x** specifies a D_floating number. For MTH$GATAND2, **x** specifies a G_floating number. For MTH$HATAND2, **x** specifies an H_floating number.

h_degrees

VMS Usage: **floating_point**
type: **H_floating**
access: **write only**
mechanism: **by reference**

Angle (in degrees) whose tangent is specified by (**y**/**x**). The **h_degrees** argument is the address of an H_floating number that is this angle. MTH$HATAND2 writes the address of the angle into **h_degrees**. The **h_degrees** argument is used only by the MTH$HATAND2 routine.

DESCRIPTION

The angle in degrees whose tangent is Y/X is computed below and where f is defined in the description of MTH$zCOSH.

Value of Input Arguments	Angle Returned
$X = 0$ or $Y/X > 2^{(f+1)}$	$90 * (\text{sign } Y)$
$X > 0$ and $Y/X =< 2^{(f+1)}$	$zATAND(Y/X)$
$X < 0$ and $Y/X =< 2^{(f+1)}$	$180 * (\text{sign } Y) + zATAND(Y/X)$

CONDITION VALUES SIGNALED

SS$_ROPRAND	Reserved operand. The MTH$xATAND2 routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.
MTH$_INVARGMAT	Invalid argument: Both **x** and **y** are zero. LIB$SIGNAL copies the floating-point reserved operand to the mechanism argument vector CHF$L_MCH_SAVR0/R1. The result is the floating-point reserved operand unless you have written a condition handler to change CHF$L_MCH_SAVR0/R1.

MTH$xATANH—Hyperbolic Arc Tangent

Given the hyperbolic tangent of an angle, MTH$xATANH returns the hyperbolic arc tangent of that angle.

FORMAT

MTH$ATANH *x*
MTH$DATANH *x*
MTH$GATANH *x*
MTH$HATANH *h_atanh ,x*

Each of the above four formats accepts as input one of the four floating-point types.

RETURNS

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating**
access: **write only**
mechanism: **by value**

The hyperbolic arc tangent of **x**. MTH$ATANH returns an F_floating number. MTH$DATANH returns a D_floating number. MTH$GATANH returns a G_floating number. Unlike the other three routines, MTH$HATANH returns the hyperbolic arc tangent by reference in the **h_atanh** argument.

ARGUMENTS

x

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating, H_floating**
access: **read only**
mechanism: **by reference**
Hyperbolic tangent of an angle. The **x** argument is the address of a floating-point number that is this hyperbolic tangent. For MTH$ATANH, **x** specifies an F_floating number. For MTH$DATANH, **x** specifies a D_floating number. For MTH$GATANH, **x** specifies a G_floating number. For MTH$HATANH, **x** specifies an H_floating number.

h_atanh

VMS Usage: **floating_point**
type: **H_floating**
access: **write only**
mechanism: **by reference**
Hyperbolic arc tangent of the hyperbolic tangent specified by **x**. The **h_atanh** argument is the address of an H_floating number that is this hyperbolic tangent. MTH$HATANH writes the address of the hyperbolic tangent into **h_atanh**. The **h_atanh** argument is used only by the MTH$HATANH routine.

DESCRIPTION

The hyperbolic arc tangent function is computed as follows:

Value of x	Value Returned
$\|X\| < 1$	$zATANH(X) = zLOG((X+1)/(X-1)) /2$
$\|X\| >= 1$	An invalid argument is signaled

CONDITION VALUES SIGNALED

SS$_ROPRAND
: Reserved operand. The MTH$xATANH routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.

MTH$_INVARGMAT
: Invalid argument: $|X| >= 1$. LIB$SIGNAL copies the floating-point reserved operand to the mechanism argument vector CHF$L_MCH_SAVR0/R1. The result is the floating-point reserved operand unless you have written a condition handler to change CHF$L_MCH_SAVR0/R1.

MTH$CxABS—Complex Absolute Value

MTH$CxABS returns the absolute value of a complex number (r,i).

FORMAT

MTH$CABS *complex-number*
MTH$CDABS *complex-number*
MTH$CGABS *complex-number*

Each of the above three formats accepts as input one of the three floating-point complex types.

RETURNS

VMS Usage:	**floating_point**
type:	**F_floating, D_floating, G_floating**
access:	**write only**
mechanism:	**by value**

The absolute value of a complex number. MTH$CABS returns an F_floating number. MTH$CDABS returns a D_floating number. MTH$CGABS returns a G_floating number.

ARGUMENT

complex-number

VMS Usage:	**complex_number**
type:	**F_floating complex, D_floating complex, G_floating complex**
access:	**read only**
mechanism:	**by reference**

A complex number (r,i), where r and i are both floating-point complex values. The **complex-number** argument is the address of this complex number. For MTH$CABS, **complex-number** specifies an F_floating complex number. For MTH$CDABS, **complex-number** specifies a D_floating complex number. For MTH$CGABS, **complex-number** specifies a G_floating complex number.

DESCRIPTION

The complex absolute value is computed as follows where *MAX* is the larger of |r| and |i|, and *MIN* is the smaller of |r| and |i|.

```
result = MAX * SQRT((MIN/MAX)^2 + 1)
```

CONDITION VALUES SIGNALED

SS$_ROPRAND	Reserved operand. The MTH$CxABS routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.
MTH$_FLOOVEMAT	Floating-point overflow in Math Library when both r and i are large.

EXAMPLES

1

```
C+
C     This FORTRAN example forms the absolute value of an
C     F_floating complex number using MTH$CABS and the
C     FORTRAN random number generator RAN.
C
C     Declare Z as a complex value and MTH$CABS as a REAL*4 value.
C     MTH$CABS will return the absolute value of Z:   Z_NEW = MTH$CABS(Z).
C-
          COMPLEX Z
          COMPLEX CMPLX
          REAL*4 Z_NEW,MTH$CABS
          INTEGER M
          M = 1234567
C+
C     Generate a random complex number with the FORTRAN generic CMPLX.
C-
          Z = CMPLX(RAN(M),RAN(M))
C+
C     Z is a complex number (r,i) with real part "r" and
C     imaginary part "i".
C-
         TYPE *, ' The complex number z is',z
         TYPE *, ' It has real part',REAL(Z),'and imaginary part',AIMAG(Z)
         TYPE *,' '
C+
C     Compute the complex absolute value of Z.
C-
          Z_NEW = MTH$CABS(Z)
          TYPE *, ' The complex absolute value of',z,' is',Z_NEW
          END
```

This example uses an F_floating complex number for **complex-number**. The output of this FORTRAN example is as follows:

```
The complex number z is (0.8535407,0.2043402)
It has real part  0.8535407    and imaginary part  0.2043402
The complex absolute value of (0.8535407,0.2043402) is  0.8776597
```

2
```
C+
C     This FORTRAN example forms the absolute
C     value of a G_floating complex number using
C     MTH$CGABS and the FORTRAN random number
C     generator RAN.
C
C     Declare Z as a complex value and MTH$CGABS as a
C     REAL*8 value. MTH$CGABS will return the absolute
C     value of Z: Z_NEW = MTH$CGABS(Z).
C-

        COMPLEX*16 Z
        REAL*8 Z_NEW,MTH$CGABS
C+
C     Generate a random complex number with the FORTRAN
C     generic CMPLX.
C-

        Z = (12.34567890123,45.536376385345)
        TYPE *, ' The complex number z is',z
        TYPE *, ' '
C+
C     Compute the complex absolute value of Z.
C-

        Z_NEW = MTH$CGABS(Z)
        TYPE *, ' The complex absolute value of',z,' is',Z_NEW
        END
```

This FORTRAN example uses a G_floating complex number for **complex-number**. Because this example uses a G_floating number, it must be compiled as follows:

```
$ FORTRAN/G MTHEX.FOR
```

Notice the difference in the precision of the output generated:

```
The complex number z is (12.3456789012300,45.5363763853450)
The complex absolute value of (12.3456789012300,45.5363763853450) is
 47.1802645376230
```

MTH$CxCOS—Complex Cosine

MTH$CxCOS returns the complex cosine of a complex number.

FORMAT

MTH$CCOS *complex-number*
MTH$CDCOS *complex-cosine ,complex-number*
MTH$CGCOS *complex-cosine ,complex-number*

Each of the above three formats accepts as input one of the three floating-point types.

RETURNS

VMS Usage: **complex_number**
type: **F_floating complex**
access: **write only**
mechanism: **by value**

The complex cosine of the complex input number. MTH$CCOS returns an F_floating complex number. MTH$CDCOS returns a D_floating complex number by reference in the **complex-cosine** argument. MTH$CGCOS returns a G_floating complex number by reference in the **complex-cosine** argument.

ARGUMENTS

complex-number

VMS Usage: **complex_number**
type: **F_floating complex, D_floating complex, G_floating complex**
access: **read only**
mechanism: **by reference**

A complex number (r,i) where r and i are floating-point numbers. The **complex-number** argument is the address of this complex number. For MTH$CCOS, **complex-number** specifies an F_floating complex number. For MTH$CDCOS, **complex-number** specifies a D_floating complex number. For MTH$CGCOS, **complex-number** specifies a G_floating complex number.

complex-cosine

VMS Usage: **complex_number**
type: **D_floating complex, G_floating complex**
access: **write only**
mechanism: **by reference**

Complex cosine of the **complex-number**. The complex cosine routines that have D_floating and G_floating complex input values write the address of the complex cosine into the **complex-cosine** argument. For MTH$CDCOS, the **complex-cosine** argument specifies a D_floating complex number. For MTH$CGCOS, the **complex-number** argument specifies a G_floating complex number. For MTH$CCOS, **complex-number** is not used.

DESCRIPTION

The complex cosine is calculated as follows:

```
result = (COS(r) * COSH(i), -SIN(r) * SINH(i))
```

CONDITION VALUES SIGNALED

SS$_ROPRAND	Reserved operand. The MTH$CxCOS routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.
MTH$_FLOOVEMAT	Floating-point overflow in Math Library: the absolute value of **i** is greater than about 88.029 for F_floating and D_floating values or greater than 709.089 for G_floating values.

EXAMPLES

1

```
C+
C     This FORTRAN example forms the complex
C     cosine of an F_floating complex number using
C     MTH$CCOS and the FORTRAN random number
C     generator RAN.
C
C     Declare Z and MTH$CCOS as complex values.
C     MTH$CCOS will return the cosine value of
C     Z:           Z_NEW = MTH$CCOS(Z)
C-

        COMPLEX Z,Z_NEW,MTH$CCOS
        COMPLEX CMPLX
        INTEGER M
        M = 1234567
C+
C     Generate a random complex number with the
C     FORTRAN generic CMPLX.
C-

        Z = CMPLX(RAN(M),RAN(M))
C+
C     Z is a complex number (r,i) with real part "r" and
C     imaginary part "i".
C-

        TYPE *, ' The complex number z is',z
        TYPE *, ' It has real part',REAL(Z),'and imaginary part',AIMAG(Z)
        TYPE *, ' '
C+
C     Compute the complex cosine value of Z.
C-

        Z_NEW = MTH$CCOS(Z)
        TYPE *, ' The complex cosine value of',z,' is',Z_NEW
        END
```

This FORTRAN example demonstrates the use of MTH$CxCOS, using the MTH$CCOS entry point. The output of this program is as follows:

```
The complex number z is (0.8535407,0.2043402)
It has real part  0.8535407    and imaginary part  0.2043402
The complex cosine value of (0.8535407,0.2043402) is (0.6710899,-0.1550672)
```

2

```
C+
C     This FORTRAN example forms the complex
C     cosine of a D_floating complex number using
C     MTH$CDCOS and the FORTRAN random number
C     generator RAN.
C
C     Declare Z and MTH$CDCOS as complex values.
C     MTH$CDCOS will return the cosine value of
C     Z:         Z_NEW = MTH$CDCOS(Z)
C-

        COMPLEX*16 Z,Z_NEW,MTH$CDCOS
        COMPLEX*16 DCMPLX
        INTEGER M
        M = 1234567

C+
C     Generate a random complex number with the
C     FORTRAN generic DCMPLX.
C-

        Z = DCMPLX(RAN(M),RAN(M))

C+
C     Z is a complex number (r,i) with real part "r" and
C     imaginary part "i".
C-

        TYPE *, ' The complex number z is',z
        TYPE *, ' '

C+
C     Compute the complex cosine value of Z.
C-

        Z_NEW = MTH$CDCOS(Z)
        TYPE *, ' The complex cosine value of',z,' is',Z_NEW
        END
```

This FORTRAN example program demonstrates the use of MTH$CxCOS, using the MTH$CDCOS entry point. Notice the high precision of the output generated:

```
The complex number z is (0.8535407185554504,0.2043401598930359)
The complex cosine value of (0.8535407185554504,0.2043401598930359) is
(0.6710899028500762,-0.1550672019621661)
```

MTH$CxEXP—Complex Exponential

MTH$CxEXP returns the complex exponential of a complex number.

FORMAT

MTH$CEXPP *complex-number*
MTH$CDEXP *complex-exp ,complex-number*
MTH$CGEXP *complex-exp ,complex-number*

Each of the above three formats accepts as input one of the three floating-point complex types.

RETURNS

VMS Usage: **complex_number**
type: **F_floating complex**
access: **write only**
mechanism: **by value**

Complex exponential of the complex input number. MTH$CEXP returns an F_floating complex number. MTHE$CDEXP returns a D_floating complex number by reference in the **complex-exp** argument. MTH$CGEXP returns a G_floating complex number in the **complex-exp** argument.

ARGUMENTS

complex-number

VMS Usage: **complex_number**
type: **F_floating complex, D_floating complex, G_floating complex**
access: **read only**
mechanism: **by reference**

Complex number whose complex exponential is to be returned. This complex number has the form (r,i), where "r" is the real part and "i" is the imaginary part. The **complex-number** argument is the address of this complex number. For MTH$CEXP, **complex-number** specifies an F_floating number. For MTH$CDEXP, **complex-number** specifies a D_floating number. For MTH$CGEXP, **complex-number** specifies a G_floating number.

complex-exp

VMS Usage: **complex_number**
type: **D_floating complex, G_floating complex**
access: **write only**
mechanism: **by reference**

Complex exponential of **complex-number**. The complex exponential routines that have D_floating complex and G_floating complex input values write the complex-exp into this argument. For MTH$CDEXP, **complex-exp** argument specifies a D_floating complex number. For MTH$CGEXP, **complex-exp** specifies a G_floating complex number. For MTH$CCEXP, **complex-exp** is not used.

DESCRIPTION

The complex exponential is computed as follows:

```
complex-exp = (EXP(r)*COS(i), EXP(r)*SIN(i))
```

CONDITION VALUES SIGNALED

SS$_ROPRAND	Reserved operand. The MTH$CxEXP routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.
MTH$_FLOOVEMAT	Floating-point overflow in Math Library: the absolute value of r is greater than about 88.029 for F_floating and D_floating values or greater than about 709.089 for G_floating values.

EXAMPLES

1

```
C+
C     This FORTRAN example forms the complex exponential
C     of an F_floating complex number using MTH$CEXP
C     and the FORTRAN random number generator RAN.
C
C     Declare Z and MTH$CEXP as complex values. MTH$CEXP
C     will return the exponential value of Z: Z_NEW = MTH$CEXP(Z)
C-

        COMPLEX Z,Z_NEW,MTH$CEXP
        COMPLEX CMPLX
        INTEGER M
        M = 1234567

C+
C     Generate a random complex number with the
C     FORTRAN generic CMPLX.
C-

        Z = CMPLX(RAN(M),RAN(M))

C+
C     Z is a complex number (r,i) with real part "r"
C     and imaginary part "i".
C-

        TYPE *, ' The complex number z is',z
        TYPE *, ' It has real part',REAL(Z),'and imaginary part',AIMAG(Z)
        TYPE *, ' '

C+
C     Compute the complex exponential value of Z.
C-

        Z_NEW = MTH$CEXP(Z)
        TYPE *, ' The complex exponential value of',z,' is',Z_NEW
        END
```

This FORTRAN program demonstrates the use of MTH$CEXP as a function call. The output generated by this example is as follows:

```
The complex number z is (0.8535407,0.2043402)
It has real part  0.8535407    and imaginary part  0.2043402
The complex exponential value of (0.8535407,0.2043402) is (2.299097,0.4764476)
```

2
```
C+
C    This FORTRAN example forms the complex exponential
C    of a G_floating complex number using MTH$CGEXP
C    and the FORTRAN random number generator RAN.
C
C    Declare Z and MTH$CGEXP as complex values.
C    MTH$CGEXP will return the exponential value
C    of Z:       CALL MTH$CGEXP(Z_NEW,Z)
C-

        COMPLEX*16 Z,Z_NEW
        COMPLEX*16 MTH$GCMPLX
        REAL*8 R,I
        INTEGER M
        M = 1234567
C+
C    Generate a random complex number with the FORTRAN
C-   generic CMPLX.
C-

        R = RAN(M)
        I = RAN(M)
        Z = MTH$GCMPLX(R,I)
        TYPE *, ' The complex number z is',z
        TYPE *, ' '
C+
C    Compute the complex exponential value of Z.
C-

        CALL MTH$CGEXP(Z_NEW,Z)
        TYPE *, ' The complex exponential value of',z,' is',Z_NEW
        END
```

This FORTRAN example demonstrates how to access MTH$CGEXP as a procedure call. Because G_floating numbers are used, this program must be compiled using the command "FORTRAN/G filename".

Notice the high precision of the output generated:

```
 The complex number z is (0.853540718555450,0.204340159893036)
 The complex exponential value of (0.853540718555450,0.204340159893036) is
(2.29909677719458,0.476447678044977)
```

MTH$CxLOG—Complex Natural Logarithm

MTH$CxLOG returns the complex natural logarithm of a complex number.

FORMAT

MTH$CLOG *complex-number*
MTH$CDLOG *complex-natlog ,complex-number*
MTH$CGLOG *complex-natlog ,complex-number*

Each of the above three formats accepts as input one of the three floating-point complex types.

RETURNS

VMS Usage: **complex_number**
type: **F_floating complex**
access: **write only**
mechanism: **by value**

The complex natural logarithm of a complex number. MTH$CLOG returns an F_floating complex number. MTH$CDLOG returns a D_floating complex-number by reference in the **complex-natlog** argument. MTH$CGLOG returns a G_floating complex number by reference in the **complex-natlog** argument.

ARGUMENTS

complex-number

VMS Usage: **complex_number**
type: **F_floating complex, D_floating complex, G_floating complex**
access: **read only**
mechanism: **by reference**
Complex number whose complex natural logarithm is to be returned. This complex number has the form (r,i), where "r" is the real part and "i" is the imaginary part. The **complex-number** argument is the address of this complex number. For MTH$CLOG, **complex-number** specifies an F_floating number. For MTH$CDLOG, **complex-number** specifies a D_floating number. For MTH$CGLOG, **complex-number** specifies a G_floating number.

complex-natlog

VMS Usage: **complex_number**
type: **D_floating complex, G_floating complex**
access: **write only**
mechanism: **by reference**
Natural logarithm of the complex number specified by **complex-number**. The complex natural logarithm routines that have D_floating complex and G_floating complex input values write the address of the complex natural logarithm into **complex-natlog**. For MTH$CDLOG, the **complex-natlog** argument specifies a D_floating complex number. For MTH$CGLOG, the **complex-natlog** argument specifies a G_floating complex number. For MTH$CLOG, **complex-natlog** is not used.

DESCRIPTION

The complex natural logarithm is computed as follows:

```
CLOG(x) = (LOG(CABS(x)), ATAN2(i,r))
```

CONDITION VALUE SIGNALED

SS$_ROPRAND	Reserved operand. The MTH$CxLOG routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.

EXAMPLES

1

```
C+
C       This FORTRAN example forms the complex logarithm
C       of a D_floating complex number by using MTH$CDLOG
C       and the FORTRAN random number generator RAN.
C
C       Declare Z and MTH$CDLOG as complex values. Then MTH$CDLOG
C       will return the logarithm of Z: CALL MTH$CDLOG(Z_NEW,Z).
C
C       Declare Z,Z_LOG, and MTH$DCMPLX as complex values,
C       and R and I as real values. MTH$DCMPLX takes two real
C       arguments and returns one complex number.
C
C       Given a complex number Z, MTH$CDLOG(Z) returns the
C       complex natural logarithm of Z.
C-
        COMPLEX*16 Z,Z_NEW,MTH$DCMPLX
        REAL*8 R,I
        R = 3.1425637846746565
        I = 7.43678469887
        Z = MTH$DCMPLX(R,I)
C+
C       Z is a complex number (r,i) with real part "r" and imaginary
C       part "i".
C-
        TYPE *, ' The complex number z is',z
        TYPE *, ' '
        CALL MTH$CDLOG(Z_NEW,Z)
        TYPE *,' The complex logarithm of',z,' is',Z_NEW
        END
```

This FORTRAN example program uses MTH$CDLOG by calling it as a procedure. The output generated by this program is as follows:

```
The complex number z is (3.142563784674657,7.436784698870000)
The complex logarithm of (3.142563784674657,7.436784698870000) is
(2.088587642177504,1.170985519274141)
```

2

Additional examples of using MTH$CLOG from VAX MACRO (using both the CALLS and the CALLG instructions) appear in Section 4.7.

MTH$xCMPLX—Complex Number Made from Floating-Point

MTH$xCMPLX returns a complex number from two floating-point input values.

FORMAT

MTH$CMPLX *real-part ,imag-part*
MTH$DCMPLX *complx ,real-part ,imag-part*
MTH$GCMPLX *complx ,real-part ,imag-part*

Each of the above three formats accepts as input one of three floating-point types.

RETURNS

VMS Usage:	**complex_number**
type:	**F_floating complex number**
access:	**write only**
mechanism:	**by value**

A complex number. MTH$CMPLX returns an F_floating complex number. MTH$DCMPLX returns a D_floating complex number by reference in the **complx** argument. MTH$GCMPLX returns a G_floating complex number by reference in the **complx** argument.

ARGUMENTS

real-part

VMS Usage:	**floating_point**
type:	**F_floating, D_floating, G_floating**
access:	**write only**
mechanism:	**by reference**

Real part of a complex number. The **real-part** argument is the address of a floating-point number that contains this real part, r of (r,i). For MTH$CMPLX, **real-part** specifies an F_floating number. For MTH$DCMPLX, **real-part** specifies a D_floating number. For MTH$GCMPLX, **real-part** specifies a G_floating number.

imag-part

VMS Usage:	**floating_point**
type:	**F_floating, D_floating, G_floating**
access:	**write only**
mechanism:	**by reference**

Imaginary part of a complex number. The **imag-parg** argument is the address of a floating-point number that contains this imaginary part, i of (r,i). For MTH$CMPLX, **imag-part** specifies an F_floating number. For MTH$DCMPLX, **imag-part** specifies a D_floating number. For MTH$GCMPLX, **imag-part** specifies a G_floating number.

complx

VMS Usage: **complex_number**
type: **D_floating complex, G_floating complex**
access: **write only**
mechanism: **by reference**

The floating-point complex value of a complex number. The complex exponential functions that have D_floating complex and G_floating complex input values write the address of this floating-point complex value into **complx**. For MTH$DCMPLX, **complx** specifies a D_floating complex number. For MTH$GCMPLX, **complx** specifies a G_floating complex number. For MTH$CMPLX, **complx** is not used.

CONDITION VALUE SIGNALED

SS$_ROPRAND	Reserved operand. The MTH$xCMPL routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.

EXAMPLES

1

```
C+
C    This FORTRAN example forms two F_floating
C    point complex numbers using MTH$CMPLX
C    and the FORTRAN random number generator RAN.
C
C    Declare Z and MTH$CMPLX as complex values, and R
C    and I as real values.  MTH$CMPLX takes two real
C    F_floating point values and returns one COMPLEX*8 number.
C
C    Note, since CMPLX is a generic name in FORTRAN, it would be sufficient
C    to use CMPLX. CMPLX must be declare to be of type COMPLEX*8.
C
C    Z = CMPLX(R,I)
C-

        COMPLEX Z,MTH$CMPLX,CMPLX
        REAL*4 R,I
        INTEGER M
        M = 1234567
        R = RAN(M)
        I = RAN(M)
        Z = MTH$CMPLX(R,I)
C+
C    Z is a complex number (r,i) with real part "r" and
C    imaginary part "i".
C-
        TYPE *, ' The two input values are:',R,I
        TYPE *, ' The complex number z is',z
        z = CMPLX(RAN(M),RAN(M))
        TYPE *, ' '
        TYPE *, ' Using the FORTRAN generic CMPLX with random R and I:'
        TYPE *, ' The complex number z is',z
        END
```

This FORTRAN example program demonstrates the use of MTH$CMPLX. The output generated by this program is as follows:

```
The two input values are:  0.8535407      0.2043402
The complex number z is (0.8535407,0.2043402)
Using the FORTRAN generic CMPLX with random R and I:
The complex number z is (0.5722565,0.1857677)
```

2

```
C+
C    This FORTRAN example forms two D_floating
C    point complex numbers using MTH$CMPLX
C    and the FORTRAN random number generator RAN.
C
C    Declare Z and MTH$DCMPLX as complex values, and R
C    and I as real values.  MTH$DCMPLX takes two real
C    D_floating point values and returns one
C    COMPLEX*16 number.
C
C-
        COMPLEX*16 Z
        REAL*8 R,I
        INTEGER M
        M = 1234567
        R = RAN(M)
        I = RAN(M)
        CALL MTH$DCMPLX(Z,R,I)
C+
C    Z is a complex number (r,i) with real part "r" and imaginary
C    part "i".
C-
        TYPE *, ' The two input values are:',R,I
        TYPE *, ' The complex number z is',Z
        END
```

This FORTRAN example demonstrates how to make a procedure call to MTH$DCMPLX. Notice the difference in the precision of the output generated.

```
The two input values are:  0.8535407185554504      0.2043401598930359
The complex number z is (0.8535407185554504,0.2043401598930359)
```

MTH$xCONJG—Conjugate of a Complex Number

MTH$xCONJG returns the complex conjugate (r,-i) of a complex number (r,i).

FORMAT

MTH$CONJG *complex-number*
MTH$DCONJG *complex-conjugate ,complex-number*
MTH$GCONJG *complex-conjugate ,complex-number*

Each of the above three formats accepts as input one of the three floating-point complex types.

RETURNS

VMS Usage: **complex_number**
type: **F_floating complex**
access: **write only**
mechanism: **by value**

Complex conjugate of a complex number. MTH$CONJG returns an F_floating complex number. MTH$DCONJG returns a D_floating complex number by reference in the **complex-conjugate** argument. MTH$GCONJG returns a G_floating complex number by reference in the **complex-conjugate** argument.

ARGUMENTS

complex-number

VMS Usage: **complex_number**
type: **F_floating complex, D_floating complex, G_floating complex**
access: **read only**
mechanism: **by reference**

A complex number (r,i), where r and i are floating-point numbers. The **complex-number** argument is the address of this floating-point complex number. For MTH$CONJG, **complex-number** specifies an F_floating number. For MTH$DCONJG, **complex-number** specifies a D_floating number. For MTH$GCONJG, **complex-number** specifies a G_floating number.

complex-conjugate

VMS Usage: **complex_number**
type: **D_floating complex, G_floating complex**
access: **write only**
mechanism: **by reference**

The complex conjugate (r,-i) of the complex number specified by **complex-number**. MTH$DCONJG and MTH$GCONJG write the address of this complex conjugate into **complex-conjugate**. For MTH$DCONJG, the **complex-conjugate** argument specifies the address of a D_floating complex number. For MTH$GCONJG, the **complex-conjugate** argument specifies

the address of a G_floating complex number. For MTH$CONJG, **complex-conjugate** is not used.

CONDITION VALUE SIGNALED

SS$_ROPRAND — Reserved operand. The MTH$xCONJG routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.

EXAMPLE

```
C+
C       This FORTRAN example forms the complex conjugate
C       of a G_floating complex number using MTH$GCONJG
C       and the FORTRAN random number generator RAN.
C
C       Declare Z, Z_NEW, and MTH$GCONJG as a complex values.
C       MTH$GCONJG will return the complex conjugate
C       value of Z:   Z_NEW = MTH$GCONJG(Z).
C-
          COMPLEX*16 Z,Z_NEW,MTH$GCONJG
          COMPLEX*16 MTH$GCMPLX
          REAL*8 R,I,MTH$GREAL,MTH$GIMAG
          INTEGER M
          M = 1234567
C+
C       Generate a random complex number with the
C       FORTRAN generic CMPLX.
C-
          R = RAN(M)
          I = RAN(M)
          Z = MTH$GCMPLX(R,I)
          TYPE *, ' The complex number z is',z
          TYPE 1,MTH$GREAL(Z),MTH$GIMAG(Z)
    1     FORMAT(' with real part ',F20.16,' and imaginary part',F20.16)
          TYPE *, ' '
C+
C       Compute the complex absolute value of Z.
C-
          Z_NEW = MTH$GCONJG(Z)
          TYPE *, ' The complex conjugate value of',z,' is',Z_NEW
          TYPE 1,MTH$GREAL(Z_NEW),MTH$GIMAG(Z_NEW)
          END
```

This FORTRAN example demonstrates how to make a function call to MTH$GCONJG. Because G_floating numbers are used, the examples must be compiled with the statement "FORTRAN/G filename".

The output generated by this program is as follows:

```
 The complex number z is (0.853540718555450,0.204340159893036)
with real part   0.8535407185554504 and imaginary part  0.2043401598930359
 The complex conjugate value of (0.853540718555450,0.204340159893036) is
(0.853540718555450,-0.204340159893036)
with real part   0.8535407185554504 and imaginary part -0.2043401598930359
```

MTH$xCOS—Cosine of Angle Expressed in Radians

MTH$xCOS returns the cosine of a given angle (in radians).

FORMAT

MTH$COS *x*
MTH$DCOS *x*
MTH$GCOS *x*
MTH$HCOS *h_cosine ,x*

Each of the above four formats accepts as input one of the four floating-point types.

jsb entries

MTH$COS_R4
MTH$DCOS_R7
MTH$GCOS_R7
MTH$HCOS_R5

Each of the above four JSB entries accepts as input one of the four floating-point types.

RETURNS

VMS Usage:	**floating_point**
type:	**F_floating, D_floating, G_floating**
access:	**write only**
mechanism:	**by value**

Cosine of the angle. MTH$COS returns an F_floating number. MTH$DCOS returns a D_floating number. MTH$GCOS returns a G_floating number. Unlike the other three routines, MTH$HCOS returns the cosine by reference in the h_cosine argument.

ARGUMENTS

x

VMS Usage:	**floating_point**
type:	**F_floating, D_floating, G_floating, H_floating**
access:	**read only**
mechanism:	**by reference**

The angle in radians. The **x** argument is the address of a floating-point number. For MTH$COS, **x** is an F_floating number. For MTH$DCOS, **x** specifies a D_floating number. For MTH$GCOS, **x** specifies a G_floating number. For MTH$HCOS, **x** specifies an H_floating number.

h_cosine

VMS Usage: **floating_point**
type: **H_floating**
access: **write only**
mechanism: **by reference**

Cosine of the angle specified by **x**. The **h_cosine** argument is the address of an H_floating number that is this cosine. MTH$HCOS writes the address of the cosine into **h_cosine**. The **h_cosine** argument is used only by the MTH$HCOS routine.

DESCRIPTION

See the MTH$xSINCOS routine for the algorithm which is used to compute the cosine.

CONDITION VALUE SIGNALED

SS$_ROPRAND	Reserved operand. The MTH$xCOS procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.

MTH$xCOSD—Cosine of Angle Expressed in Degrees

MTH$xCOSD returns the cosine of a given angle (in degrees).

FORMAT

MTH$COSD *x*
MTH$DCOSD *x*
MTH$GCOSD *x*
MTH$HCOSD *h_cosine ,x*

Each of the above four formats accepts as input one of the four floating-point types.

jsb entries

MTH$COSD_R4
MTH$DCOSD_R7
MTH$GCOSD_R7
MTH$HCOSD_R5

Each of the above four JSB entries accepts as input one of the four floating-point types.

RETURNS

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating**
access: **write only**
mechanism: **by value**

Cosine of the angle. MTH$COSD returns an F_floating number. MTH$DCOSD returns a D_floating number. MTH$GCOSD returns a G_floating number. Unlike the other three routines, MTH$HCOSD returns the angle by reference in the **h_degrees** argument.

ARGUMENTS

x

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating, H_floating**
access: **read only**
mechanism: **by reference**

Angle (in degrees). The **x** argument is the address of a floating-point number. For MTH$COSD, **x** specifies an F_floating number. For MTH$DCOSD, **x** specifies a D_floating number. For MTH$GCOSD, **x** specifies a G_floating number. For MTH$HCOSD, **x** specifies an H_floating number.

h_cosine

VMS Usage: **floating_point**
type: **H_floating**
access: **write only**
mechanism: **by reference**

Cosine of the angle specified by **x**. The **h_cosine** argument is the address of an H_floating number that is this cosine. MTH$HCOSD writes this cosine into **h_cosine**. The **h_cosine** argument is used only by the MTH$HCOSD routine.

DESCRIPTION

See the MTH$SINCOSD routine for the algorithm which is used to compute the cosine.

CONDITION VALUE SIGNALED

SS$_ROPRAND	Reserved operand. The MTH$xCOSD procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.

MTH$xCOSH—Hyperbolic Cosine

MTH$xCOSH returns the hyperbolic cosine of the input value.

FORMAT

MTH$COSH *x*
MTH$DCOSH *x*
MTH$GCOSH *x*
MTH$HCOSH *h_cosh ,x*

Each of the above four formats accepts as input one of the four floating-point types.

RETURNS

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating**
access: **write only**
mechanism: **by value**

The hyperbolic cosine of the input value **x**. MTH$COSH returns an F_floating number. MTH$DCOSH returns a D_floating number. MTH$GCOSH returns a G_floating number. Unlike the other three routines, MTH$HCOSH returns the hyperbolic cosine by reference in the **h_cosh** argument.

ARGUMENTS

x

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating, H_floating**
access: **read only**
mechanism: **by reference**
The input value. The **x** argument is the address of this input value. For MTH$COSH, **x** specifies an F_floating number. For MTH$DCOSH, **x** specifies a D_floating number. For MTH$GCOSH, **x** specifies a G_floating number. For MTH$HCOSH, **x** specifies an H_floating number.

h_cosh

VMS Usage: **floating_point**
type: **H_floating**
access: **write only**
mechanism: **by reference**
Hyperbolic cosine of the input value specified by **x**. The **h_cosh** argument is the address of an H_floating number that is this hyperbolic cosine. MTH$HCOSH writes the address of the hyperbolic cosine into **h_cosh**. The **h_cosh** argument is used only by the MTH$HCOSH routine.

DESCRIPTION

Computation of the hyperbolic cosine depends on the magnitude of the input argument. The range of the function is partitioned using four data-type-dependent constants: a(z), b(z), c(z), and d(z). The subscript *z* indicates the data type. The constants depend on the number of exponent bits (*e*) and the number of fraction bits (*f*) associated with the data type (z).

The values of *e* and *f* are:

z	e	f
F	8	24
D	8	56
G	11	53
H	15	113

The values of the constants in terms of *e* and *f* are:

Variable	Value
a(z)	$2^{(-f/2)}$
b(z)	CEILING[$(f+1)/2 \bullet ln(2)$] for F,D, and G
	$(f+1)/2 \bullet ln(2)$ for H
c(z)	$(2^{(e-1)-1}) \bullet ln(2)$
d(z)	$c(z)+ln(2)$

Based on the above definitions, zCOSH(X) is computed as follows:

Value of X	Value Returned
$\lvert X \rvert < a(z)$	1
$a(z) =< \lvert X \rvert < .25$	Computed using a power series expansion in $\lvert X \rvert^2$
$.25 =< \lvert X \rvert < b(z)$	$(zEXP(\lvert X \rvert) + 1/zEXP(\lvert X \rvert))/2$
$b(z) =< \lvert X \rvert < c(z)$	$zEXP(\lvert X \rvert)/2$
$c(z) =< \lvert X \rvert < d(z)$	$zEXP(\lvert X \rvert - ln(2))$
$d(z) =< \lvert x \rvert$	Overflow occurs

CONDITION VALUES SIGNALED

SS$_ROPRAND

Reserved operand. The MTH$xCOSH procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.

MTH$_FLOOVEMAT

Floating-point overflow in Math Library: the absolute value of **x** is greater than about *yyy*; LIB$SIGNAL copies the reserved operand to the signal mechanism vector. The result is the reserved operand -0.0 unless a condition handler changes the signal mechanism vector.

The values of *yyy* are:

MTH$COSH	88.722
MTH$DCOSH	88.722
MTH$GCOSH	709.782
MTH$HCOSH	11356.523

MTH$CxSIN—Complex Sine of Complex Number

MTH$CxSIN returns the complex sine of a complex number (r,i).

FORMAT

MTH$CSIN *complex-number*
MTH$CDSIN *complex-sine ,complex-number*
MTH$CGSIN *complex-sine ,complex-number*

Each of the above three formats accepts as input one of the three floating-point complex types.

RETURNS

VMS Usage: **complex_number**
type: **F_floating complex**
access: **write only**
mechanism: **by value**

Complex sine of the complex number. MTH$CSIN returns an F_floating complex number. MTH$CDSIN returns a D_floating complex number by reference in the **complex-sine** argument. MTH$CGSIN returns a G_floating complex number in the **complex-sine** argument.

ARGUMENTS

complex-number

VMS Usage: **complex_number**
type: **F_floating complex, D_floating complex, G_floating complex**
access: **read only**
mechanism: **by reference**

A complex number (r,i), where r and i are floating-point numbers. The **complex-number** argument is the address of this complex number. For MTH$CSIN, **complex-number** specifies an F_floating complex number. For MTH$CDSIN, **complex-number** specifies a D_floating complex number. For MTH$CGSIN, **complex-number** specifies a G_floating complex number.

complex-sine

VMS Usage: **floating_point**
type: **D_floating, G_floating**
access: **write only**
mechanism: **by reference**

Complex sine of the complex number. The complex sine routines with D_floating complex and G_floating complex input values write the address of the complex sine into this **complex-sine** argument. For MTH$CDSIN, **complex-sine** specifies a D_floating complex number. For MTH$CGSIN, **complex-sine** specifies a G_floating complex number. For MTH$CSIN, **complex-sine** is not used.

DESCRIPTION

The complex sine is computed as follows:

```
complex-sine = (SIN(r) * COSH(i), COS(r) * SINH(i))
```

CONDITION VALUES SIGNALED

SS$_ROPRAND	Reserved operand. The MTH$CxSIN procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.
MTH$_FLOOVEMAT	Floating-point overflow in Math Library: the absolute value of I is greater than about 88.029 for F_floating and D_floating values or greater than about 709.089 for G_floating values.

EXAMPLE

```
C+
C    This FORTRAN example forms the complex
C    sine of a G_floating complex number using
C    MTH$CGSIN and the FORTRAN random number
C    generator RAN.
C
C    Declare Z and MTH$CGSIN as complex values.
C    MTH$CGSIN will return the sine value
C    of Z:       CALL MTH$CGSIN(Z_NEW,Z)
C-
        COMPLEX*16 Z,Z_NEW
        COMPLEX*16 DCMPLX
        REAL*8 R,I
        INTEGER M
        M = 1234567
C+
C    Generate a random complex number with the
C    FORTRAN generic DCMPLX.
C-
        R = RAN(M)
        I = RAN(M)
        Z = DCMPLX(R,I)
C+
C     Z is a complex number (r,i) with real part "r" and
C     imaginary part "i".
C-
        TYPE *, ' The complex number z is',z
        TYPE *, ' '
C+
C    Compute the complex sine value of Z.
C-
        CALL MTH$CGSIN(Z_NEW,Z)
        TYPE *, ' The complex sine value of',z,' is',Z_NEW
        END
```

This FORTRAN examples demonstrates a procedure call to MTH$CGSIN. Because this program uses G_floating numbers, it must be compiled with the statement "FORTRAN/G filename".

The output generated by this program is as follows:

```
The complex number z is (0.853540718555450,0.204340159893036)
The complex sine value of (0.853540718555450,0.204340159893036) is
(0.769400835484975,0.1352533409122566)
```

MTH$CxSQRT—Complex Square Root

MTH$CxSQRT returns the complex square root of a complex number (r,i).

FORMAT

MTH$CSQRT *complex-number*
MTH$CDSQRT *complex-sqrt ,complex-number*
MTH$CGSQRT *complex-sqrt ,complex-number*

Each of the above three formats accepts as input one of the three floating-point complex types.

RETURNS

VMS Usage: **complex_number**
type: **F_floating complex**
access: **write only**
mechanism: **by value**

The complex square root of **complex-number**. MTH$CSQRT returns an F_floating number. MTH$CDSQRT returns a D_floating complex number by reference in the **complex-sqrt** argument. MTH$CGSQRT returns a G_floating complex number in the **complex-sqrt** argument.

ARGUMENTS

complex-number

VMS Usage: **complex_number**
type: **F_floating complex, D_floating complex, G_floating complex**
access: **read only**
mechanism: **by reference**

Complex number (r,i). The **complex-number** argument contains the address of this complex number. For MTH$CSQRT, **complex-number** specifies an F_floating number. For MTH$CDSQRT, **complex-number** specifies a D_floating number. For MTH$CGSQRT, **complex-number** specifies a G_floating number.

complex-sqrt

VMS Usage: **complex_number**
type: **D_floating complex, G_floating complex**
access: **write only**
mechanism: **by reference**

Complex square root of the complex number specified by **complex-number**. The complex square root routines that have D_floating complex and G_floating complex input values write the complex square root into **complex-sqrt**. For MTH$CDSQRT, **complex-sqrt** specifies a D_floating complex number. For MTH$CGSQRT, **complex-sqrt** specifies a G_floating complex number. For MTH$CSQRT, **complex-sqrt** is not used.

DESCRIPTION

The complex square root is computed as follows.

First, calculate **ROOT** and **Q** using the following equations:

```
ROOT = SQRT((ABS(r) + (CABS((r,i)))/2)
Q = i/(2 * ROOT)
```

Then, the complex result is given as follows:

r	i	CSQRT((r,i))
>=0	any	(ROOT,Q)
<0	>=0	(Q,ROOT)
<0	<0	(-Q,-ROOT)

CONDITION VALUE SIGNALED

SS$_ROPRAND

Reserved operand. The MTH$CxSQRT procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.

EXAMPLE

```
C+
C    This FORTRAN example forms the complex square
C    root of a D_floating complex number using
C    MTH$CDSQRT and the FORTRAN random number
C    generator RAN.
C
C    Declare Z and Z_NEW as complex values. MTH$CDSQRT
C    will return the complex square root of
C    Z:   CALL MTH$CDSQRT(Z_NEW,Z).
C-
        COMPLEX*16 Z,Z_NEW
        COMPLEX*16 DCMPLX
        INTEGER M
        M = 1234567
C+
C    Generate a random complex number with the
C    FORTRAN generic CMPLX.
C-
        Z = DCMPLX(RAN(M),RAN(M))
C+
C    Z is a complex number (r,i) with real part "r" and imaginary
C    part "i".
C-
        TYPE *, ' The complex number z is',z
        TYPE *, ' '
C+
C    Compute the complex complex square root of Z.
C-
        CALL MTH$CDSQRT(Z_NEW,Z)
        TYPE *, ' The complex square root of',z,' is',Z_NEW
        END
```

This FORTRAN example program demonstrates a procedure call to MTH$CDSQRT. The output generated by this program is as follows:

```
 The complex number z is (0.8535407185554504,0.2043401598930359)
 The complex square root of (0.8535407185554504,0.2043401598930359) is
(0.9303763973040062,0.1098158554350485)
```

MTH$CVT_x_x—Convert One Double-Precision Value

MTH$CVT_D_G and MTH$CVT_G_D convert one double-precision value to the destination data type and return the result as a function value. MTH$CVT_D_G converts a D_floating value to G_floating and MTH$CVT_G_D converts a G_floating value to a D_floating value.

FORMAT

MTH$CVT_D_G *source*
MTH$CVT_G_D *source*

RETURNS

VMS Usage: **floating_point**
type: **G_floating, D_floating**
access: **write only**
mechanism: **by value**

The converted value. MTH$CVT_D_G returns a G_floating value. MTH$CVT_G_D returns a D_floating value.

ARGUMENT

source
VMS Usage: **floating_point**
type: **D_floating, G_floating**
access: **read only**
mechanism: **by reference**
The input value to be converted. The **source** argument is the address of this input value. For MTH$CVT_D_G, the **source** argument specifies a D_floating number. For MTH$CVT_G_D, the **source** argument specifies a G_floating number.

DESCRIPTION

These procedures are designed to function like hardware conversion instructions. They fault on reserved operands. If floating-point overflow is detected, an error is signaled. If floating-point underflow is detected and floating-point underflow is enabled, an error is signaled.

CONDITION VALUES SIGNALED

SS$_ROPRAND	Reserved operand. The MTH$CVT_x_x procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.
MTH$_FLOOVEMAT	Floating-point overflow in Math Library.
MTH$_FLOUNDMAT	Floating-point underflow in Math Library.

MTH$CVT_xA_xA—Convert an Array of Double-Precision Values

MTH$CVT_DA_GA and MTH$CVT_GA_DA convert a contiguous array of double-precision values to the destination data type and return the results as an array. MTH$CVT_DA_GA converts D_floating values to G_floating and MTH$CVT_GA_DA converts G_floating values to D_floating.

FORMAT

MTH$CVT_DA_GA *source ,dest [,count]*
MTH$CVT_GA_DA *source ,dest [,count]*

RETURNS

MTH$CVT_DA_GA and MTH$CVT_GA_DA return the address of the output array to the **dest** argument.

ARGUMENTS

source

VMS Usage: **floating_point**
type: **D_floating, G_floating**
access: **read only**
mechanism: **by reference, array reference**

Input array of values to be converted. The **source** argument is the address of an array of floating-point numbers. For MTH$CVT_DA_GA, **source** specifies an array of D_floating numbers. For MTH$CVT_GA_DA, **source** specifies an array of a G_floating numbers.

dest

VMS Usage: **floating_point**
type: **G_floating, D_floating**
access: **write only**
mechanism: **by reference, array reference**

Output array of converted values. The **dest** argument is the address of an array of floating-point numbers. For MTH$CVT_DA_GA, **dest** specifies an array of G_floating numbers. For MTH$CVT_GA_DA, **dest** specifies an array of D_floating numbers.

count

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Number of array elements to be converted. The default value is 1. The **count** argument is the address of this number of elements.

DESCRIPTION

These procedures are designed to function like hardware conversion instructions. They fault on reserved operands. If floating-point overflow is detected, an error is signaled. If floating-point underflow is detected and floating-point underflow is enabled, an error is signaled.

CONDITION VALUES SIGNALED

SS$_ROPRAND	Reserved operand. The MTH$CVT_xA_xA procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.
MTH$_FLOOVEMAT	Floating-point overflow in Math Library.
MTH$_FLOUNDMAT	Floating-point underflow in Math Library.

MTH$xEXP—Exponential

MTH$xEXP returns the exponential of the input value.

FORMAT

MTH$EXP *x*
MTH$DEXP *x*
MTH$GEXP *x*
MTH$HEXP *h_exp ,x*

Each of the above four formats accepts as input one of the four floating-point types.

jsb entries

MTH$EXP_R4
MTH$DEXP_R6
MTH$GEXP_R6
MTH$HEXP_R6

Each of the above four JSB entries accepts as input one of the four floating-point types.

RETURNS

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating**
access: **write only**
mechanism: **by value**

The exponential of **x**. MTH$EXP returns an F_floating number. MTH$DEXP returns a D_floating number. MTH$GEXP returns a G_floating number. Unlike the other three routines, MTH$HEXP returns the exponential by reference in the **h_exp** argument.

ARGUMENTS

x

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating, H_floating**
access: **read only**
mechanism: **by reference**

The input value. The **x** argument is the address of a floating-point number. For MTH$EXP, **x** specifies an F_floating number. For MTH$DEXP, **x** specifies a D_floating number. For MTH$GEXP, **x** specifies a G_floating number. For MTH$HEXP, **x** specifies an H_floating number.

h_exp

VMS Usage: **floating_point**
type: **H_floating**
access: **write only**
mechanism: **by reference**

Exponential of the input value specified by **x**. The **h_exp** argument is the address of an H_floating number that is this exponential. MTH$HEXP writes

the address of the exponential into **h_exp**. The **h_exp** argument is used only by the MTH$HEXP routine.

DESCRIPTION

The exponential of **x** is computed as:

Value of x	Value Returned
$x > c(z)$	Overflow occurs
$x <= -d(z)$	0
$\lvert x \rvert < 2^{-(f+1)}$	1
Otherwise	$2^{Y} * 2^{U} * 2^{W}$

where:

$Y = INTEGER(x*ln2(E))$

$V = FRAC(x*ln2(E)) * 16$

$U = INTEGER(V)/16$

$W = FRAC(V)/16$

2^{W} = *polynomial approximation of degree 4,8,8,14 for z = F, D, G, or H.*

See also the section on the hyperbolic cosine for definitions of *f*, *c(z)*, and *d(z)*.

CONDITION VALUES SIGNALED

SS$_ROPRAND	Reserved operand. The MTH$xEXP routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.
MTH$_FLOOVEMAT	Floating-point overflow in Math Library: **x** is greater than *yyy*; LIB$SIGNAL copies the reserved operand to the signal mechanism vector. The result is the reserved operand -0.0 unless a condition handler changes the signal mechanism vector. The values of *yyy* are approximately:

MTH$EXP	88.029
MTH$DEXP	88.029
MTH$GEXP	709.089
MTH$HEXP	11355.830

MTH$_FLOUNDMAT	Floating-point underflow in Math Library: **x** is less than or equal to *yyy* and the caller (CALL or JSB) has set hardware floating-point underflow enable. The result is set to 0.0. If the caller has not enabled floating-point underflow (the default), a result of 0.0 is returned but no error is signaled.

The values of *yyy* are approximately:

MTH$EXP	-88.722
MTH$DEXP	-88.722
MTH$GEXP	-709.774
MTH$HEXP	-11356.523

EXAMPLE

```
IDENTIFICATION DIVISION.
PROGRAM-ID.    FLOATING_POINT.
*
*  Calls MTH$EXP using a Floating Point data type.
*  Calls MTH$DEXP using a Double Floating Point data type.
*
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 FLOAT_PT      COMP-1.
01 ANSWER_F      COMP-1.
01 DOUBLE_PT     COMP-2.
01 ANSWER_D      COMP-2.
PROCEDURE DIVISION.
P0.
        MOVE 12.34 TO FLOAT_PT.
        MOVE 3.456 TO DOUBLE_PT.
        CALL "MTH$EXP" USING BY REFERENCE FLOAT_PT GIVING ANSWER_F.
        DISPLAY " MTH$EXP of ", FLOAT_PT CONVERSION, " is ",
                                        ANSWER_F CONVERSION.
        CALL "MTH$DEXP" USING BY REFERENCE DOUBLE_PT GIVING ANSWER_D.
        DISPLAY " MTH$DEXP of ", DOUBLE_PT CONVERSION, " is ",
                                        ANSWER_D CONVERSION .
        STOP RUN.
```

This sample program demonstrates calls to MTH$EXP and MTH$DEXP from COBOL.

The output generated by this program is as follows:

```
MTH$EXP of  1.234000E+01 is  2.286620E+05
MTH$DEXP of  3.456000000000000E+00 is
3.168996280537917E+01
```

MTH$xIMAG—Imaginary Part of a Complex Number

MTH$xIMAG returns the imaginary part of a complex number.

FORMAT

MTH$AIMAG *complex-number*
MTH$DIMAG *complex-number*
MTH$GIMAG *complex-number*

Each of the above three formats corresponds to one of the three floating-point complex types.

RETURNS

VMS Usage:	**floating_point**
type:	**F_floating, D_floating, G_floating**
access:	**write only**
mechanism:	**by value**

Imaginary part of the input **complex-number**. MTH$AIMAG returns an F_floating number. MTH$DIMAG returns a D_floating number. MTH$GIMAG returns a G_floating number.

ARGUMENT

complex-number

VMS Usage:	**complex_number**
type:	**F_floating complex, D_floating complex, G_floating complex**
access:	**read only**
mechanism:	**by reference**

The input complex number. The **complex-number** argument is the address of this floating-point complex number. For MTH$AIMAG, **complex-number** specifies an F_floating number. For MTH$DIMAG, **complex-number** specifies a D_floating number. For MTH$GIMAG, **complex-number** specifies a G_floating number.

CONDITION VALUE SIGNALED

SS$_ROPRAND	Reserved operand. The MTH$xIMAG routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.

EXAMPLE

```
C+
C     This FORTRAN example forms the imaginary part of
C     a G_floating complex number using MTH$GIMAG
C     and the FORTRAN random number generator
C     RAN.
C
C     Declare Z as a complex value and MTH$GIMAG as
C     a REAL*8 value. MTH$GIMAG will return the imaginary
C     part of Z:   Z_NEW = MTH$GIMAG(Z).
C-
        COMPLEX*16 Z
        COMPLEX*16 DCMPLX
        REAL*8 R,I,MTH$GIMAG
        INTEGER M
        M = 1234567
C+
C     Generate a random complex number with the
C     FORTRAN generic CMPLX.
C-
        R = RAN(M)
        I = RAN(M)
        Z = DCMPLX(R,I)
C+
C     Z is a complex number (r,i) with real part "r" and
C     imaginary part "i".
C-
        TYPE *, ' The complex number z is',z
        TYPE *, ' It has imaginary part',MTH$GIMAG(Z)
        END
```

This FORTRAN example demonstrates a procedure call to MTH$GIMAG. Because this example uses G_floating numbers, it must be compiled with the statement "FORTRAN/G filename".

The output generated by this program is as follows:

```
The complex number z is (0.8535407185554504,0.2043401598930359)
It has imaginary part  0.2043401598930359
```

MTH$xLOG—Logarithm, Natural

MTH$xLOG returns the natural (base e) logarithm of the input argument.

FORMAT

MTH$ALOG *x*
MTH$DLOG *x*
MTH$GLOG *x*
MTH$HLOG *h_natlog ,x*

Each of the above four formats accepts as input one of the four floating-point types.

jsb entries

MTH$ALOG_R5
MTH$DLOG_R8
MTH$GLOG_R8
MTH$HLOG_R8

Each of the above four JSB entries accepts as input one of the four floating-point types.

RETURNS

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating**
access: **write only**
mechanism: **by value**

The natural logarithm of **x**. MTH$ALOG returns an F_floating number. MTH$DLOG returns a D_floating number. MTH$GLOG returns a G_floating number. Unlike the other three routines, MTH$HLOG returns the natural logarithm by reference in the **h_natlog** argument.

ARGUMENTS

x

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating, H_floating**
access: **read only**
mechanism: **by reference**
The input value. The **x** argument is the address of a floating-point number that is this value. For MTH$ALOG, **x** specifies an F_floating number. For MTH$DLOG, **x** specifies a D_floating number. For MTH$GLOG, **x** specifies a G_floating number. For MTH$HLOG, **x** specifies an H_floating number.

h_natlog

VMS Usage: **floating_point**
type: **H_floating**
access: **write only**
mechanism: **by reference**
Natural logarithm of **x**. The **h_natlog** argument is the address of an H_floating number that is this natural logarithm. MTH$HLOG writes the

address of this natural logarithm into **h_natlog**. The **h_natlog** argument is used only by the MTH$HLOG routine.

DESCRIPTION

Computation of the natural logarithm routine is based on the following:

```
(1)   ln(X*Y) = ln(X) + ln(Y)
(2)   ln(1+X) = X - X^2/2 + X^3/3 - X^4/4 ... for |X| < 1
(3)   ln(X) = ln(A) + 2* (V + V^3/3 + V^5/5 + V^7/7 ... )
            = ln(A) + V*p(V^2), where V = (X-A)/(X+A),
              A > 0, and p(y) = 2 * (1 + y/3 + y^2/5 ...)
For x = 2^n*f, where n is an integer and f is in the
interval of 0.5 to 1, define the following quantities:
If n >= 1, then N = n-1 and F = 2f
If n =< 0, then N = n and F = f
From (1) above it follows that:
(4) ln(X) = N*ln(2) + ln(F)
Based on the above relationships, zLOG is computed as follows:
(1) If |F-1| < 2^-5, zLOG(X) = N*zLOG(2) + W + W*p(W),
          where W = F-1.
(2) Otherwise, zLOG(X) = N*zLOG(2) + zLOG(A) + V*p(V^2),
          where V = (F-A)/(F+A) and A and zLOG(A)
          are obtained by table look up.
```

CONDITION VALUES SIGNALED

SS$_ROPRAND	Reserved operand. The MTH$xLOG procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.
MTH$_LOGZERNEG	Logarithm of zero or negative value: Argument **x** is less than or equal to 0.0. LIB$SIGNAL copies the floating-point reserved operand to the mechanism argument vector CHF$L_MCH_SAVR0/R1. The result is the floating-point reserved operand unless you have written a condition handler to change CHF$L_MCH_SAVR0/R1.

MTH$xLOG2—Logarithm, Base 2

MTH$xLOG returns the base 2 logarithm of the input value specified by **x**.

FORMAT

MTH$ALOG2 *x*
MTH$DLOG2 *x*
MTH$GLOG2 *x*
MTH$HLOG2 *h_log2 ,x*

Each of the above four formats accepts as input one of the four floating-point types.

RETURNS

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating**
access: **write only**
mechanism: **by value**

The base 2 logarithm of **x**. MTH$ALOG2 returns an F_floating number. MTH$DLOG2 returns a D_floating number. MTH$GLOG2 returns a G_floating number. Unlike the other three routines, MTH$HLOG2 returns the base 2 logarithm by reference in the **h_log2** argument.

ARGUMENTS

x

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating, H_floating**
access: **read only**
mechanism: **by reference**

The input value. The **x** argument is the address of a floating-point number that is this input value. For MTH$ALOG2, **x** specifies an F_floating number. For MTH$DLOG2, **x** specifies a D_floating number. For MTH$GLOG2, **x** specifies a G_floating number. For MTH$HLOG2, **x** specifies an H_floating number.

h_log2

VMS Usage: **floating_point**
type: **H_floating**
access: **write only**
mechanism: **by reference**

Base 2 logarithm of **x**. The **h_log2** argument is the address of an H_floating number that is this base 2 logarithm. MTH$HLOG2 writes the address of this logarithm into **h_log2**. The **h_log2** argument is used only by the MTH$HLOG2 routine.

DESCRIPTION

The base 2 logarithm function is computed as follows:

*zLOG2(X) = zLOG2(E) * zLOG(X)*

CONDITION VALUES SIGNALED

SS$_ROPRAND

Reserved operand. The MTH$xLOG2 procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.

MTH$_LOGZERNEG

Logarithm of zero or negative value: Argument **x** is less than or equal to 0.0. LIB$SIGNAL copies the floating-point reserved operand to the mechanism argument vector CHF$L_MCH_SAVR0/R1. The result is the floating-point reserved operand unless you have written a condition handler to change CHF$L_MCH_SAVR0/R1.

MTH$xLOG10—Logarithm, Common

MTH$xLOG10 returns the common (base 10) logarithm of the input argument.

FORMAT

MTH$ALOG10 *x*
MTH$DLOG10 *x*
MTH$GLOG10 *x*
MTH$HLOG10 *h_log10 ,x*

Each of the above four formats accepts as input one of the four floating-point types.

jsb entries

MTH$ALOG10_R5
MTH$DLOG10_R8
MTH$GLOG10_R8
MTH$HLOG10_R8

Each of the above four JSB entries accepts as input one of the four floating-point types.

RETURNS

VMS Usage:	**floating_point**
type:	**F_floating, D_floating, G_floating**
access:	**write only**
mechanism:	**by value**

The common logarithm of **x**. MTH$ALOG10 returns an F_floating number. MTH$DLOG10 returns a D_floating number. MTH$GLOG10 returns a G_floating number. Unlike the other three routines, MTH$HLOG10 returns the common logarithm by reference in the **h_log10** argument.

ARGUMENTS

x

VMS Usage:	**floating_point**
type:	**F_floating, D_floating, G_floating, H_floating**
access:	**read only**
mechanism:	**by reference**

The input value. The **x** argument is the address of a floating-point number. For MTH$ALOG10, **x** specifies an F_floating number. For MTH$DLOG10, **x** specifies a D_floating number. For MTH$GLOG10, **x** specifies a G_floating number. For MTH$HLOG10, **x** specifies an H_floating number.

h_log10

VMS Usage:	**floating_point**
type:	**H_floating**
access:	**write only**
mechanism:	**by reference**

Common logarithm of the input value specified by **x**. The **h_log10 argument** is the address of an H_floating number that is this common logarithm.

MTH$HLOG10 writes the address of the common logarithm into **h_log10**. The **h_log10** argument is used only by the MTH$HLOG10 routine.

DESCRIPTION

The common logarithm function is computed as follows:

*zLOG10(X) = zLOG10(E) * zLOG(X)*

CONDITION VALUES SIGNALED

SS$_ROPRAND	Reserved operand. The MTH$xLOG10 procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.
MTH$_LOGZERNEG	Logarithm of zero or negative value: Argument **x** is less than or equal to 0.0. LIB$SIGNAL copies the floating-point reserved operand to the mechanism argument vector CHF$L_MCH_SAVR0/R1. The result is the floating-point reserved operand unless you have written a condition handler to change CHF$L_MCH_SAVR0/R1.

MTH$RANDOM—Random-Number Generator, Uniformly Distributed

MTH$RANDOM is a general random-number generator.

FORMAT

MTH$RANDOM *seed*

RETURNS

VMS Usage: **floating_point**
type: **F_floating**
access: **write only**
mechanism: **by value**

MTH$RANDOM returns an F_floating random number.

ARGUMENT

seed
VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**

The integer seed, a 32-bit number whose high-order 24 bits are converted by MTH$RANDOM to an F_floating random number. The **seed** argument is the address of an unsigned longword that contains this integer seed. The seed is modified by each call to MTH$RANDOM.

DESCRIPTION

This routine must be called again to obtain the next pseudorandom number. The seed is updated automatically.

The result is a floating-point number that is uniformly distributed between 0.0 inclusively and 1.0 exclusively.

There are no restrictions on the seed, although it should be initialized to different values on separate runs in order to obtain different random sequences. MTH$RANDOM uses the following method to update the seed passed as the argument:

$$SEED = (69069 * SEED + 1) \quad (\text{modulo } 2^{32})$$

CONDITION VALUE SIGNALED

SS$_ROPRAND	Reserved operand. The MTH$RANDOM procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.

EXAMPLE

```
RAND:   PROCEDURE OPTIONS (MAIN);
DECLARE FOR$SECNDS ENTRY (FLOAT BINARY (24))
                RETURNS (FLOAT BINARY (24));
DECLARE MTH$RANDOM ENTRY (FIXED BINARY (31))
                RETURNS (FLOAT BINARY (24));
DECLARE TIME FLOAT BINARY (24);
DECLARE SEED FIXED BINARY (31);
DECLARE I FIXED BINARY (7);
DECLARE RESULT FIXED DECIMAL (2);
        /* Get floating random time value       */
TIME = FOR$SECNDS (0E0);
        /* Convert to fixed                     */
SEED = TIME;
        /* Generate 100 random numbers between 1 and 10 */
DO I = 1 TO 100;
        RESULT = 1 + FIXED ( (10E0 * MTH$RANDOM (SEED) ),31 );
        PUT LIST (RESULT);
        END;
END RAND;
```

This PL/I program demonstrates the use of MTH$RANDOM. The value returned by FOR$SECNDS is used as the seed for the random-number generator to insure a different sequence each time the program is run. The random value returned is scaled so as to represent values between 1 and 10.

Because this program generates random numbers, the output generated will be different each time the program is executed. One example of the outut generated by this program is as follows:

```
   7     4     6     5     9    10     5     5     3     8     8     1     3     1     3     2
   4     4     2     4     4     8     3     8     9     1     7     1     8     6     9    10
   1    10    10     6     7     3     2     2     1     2     6     6     3     9     5     8
   6     2     3     6    10     8     5     5     4     2     8     5     9     6     4     2
   8     5     4     9     8     7     6     6     8    10     9     5     9     4     5     7
   1     2     2     3     6     5     2     3     4     4     8     9     2     8     5     5
   3     8     1     5
```

MTH$xREAL—Real Part of a Complex Number

MTH$xREAL returns the real part of a complex number.

FORMAT

MTH$REAL *complex-number*
MTH$DREAL *complex-number*
MTH$GREAL *complex-number*

Each of the above three formats accepts as input one of the three floating-point complex types.

RETURNS

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating**
access: **write only**
mechanism: **by value**

Real part of the complex number. MTH$REAL returns an F_floating number. MTH$DREAL returns a D_floating number. MTH$GREAL returns a G_floating number.

ARGUMENT

complex-number
VMS Usage: **complex_number**
type: **F_floating complex, D_floating complex, G_floating complex**
access: **read only**
mechanism: **by reference**
The complex number whose real part is returned by MTH$REAL. The **complex-number** argument is the address of this floating-point complex number. For MTH$REAL, **complex-number** is an F_floating complex number. For MTH$DREAL, **complex-number** is a D_floating complex number. For MTH$GREAL, **complex-number** is a G_floating complex number.

CONDITION VALUE SIGNALED

SS$_ROPRAND	Reserved operand. The MTH$xREAL procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.

EXAMPLE

```
C+
C    This FORTRAN example forms the real
C    part of an F_floating complex number using
C    MTH$REAL and the FORTRAN random number
C    generator RAN.
C
C    Declare Z as a complex value and MTH$REAL as a
C    REAL*4 value.  MTH$REAL will return the real
C    part of Z:    Z_NEW = MTH$REAL(Z).
C-
        COMPLEX Z
        COMPLEX CMPLX
        REAL*4 MTH$REAL
        INTEGER M
        M = 1234567
C+
C    Generate a random complex number with the FORTRAN
C    generic CMPLX.
C-
        Z = CMPLX(RAN(M),RAN(M))
C+
C    Z is a complex number (r,i) with real part "r" and imaginary
C    part "i".
C-
        TYPE *, ' The complex number z is',z
        TYPE *, ' It has real part',MTH$REAL(Z)
        END
```

This FORTRAN example demonstrates the use of MTH$REAL. The output of this program is as follows:

```
The complex number z is (0.8535407,0.2043402)
It has real part  0.8535407
```

MTH$xSIN—Sine of Angle Expressed in Radians

MTH$xSIN returns the sine of a given angle (in radians).

FORMAT

MTH$SIN *x*
MTH$DSIN *x*
MTH$GSIN *x*
MTH$HSIN *h_sine ,x*

Each of the above four formats accepts as input one of the four floating-point types.

jsb entries

MTH$SIN_R4
MTH$DSIN_R7
MTH$GSIN_R7
MTH$HSIN_R5

Each of the above four JSB entries accepts as input one of the four floating-point types.

RETURNS

VMS Usage:	**floating_point**
type:	**F_floating, D_floating, G_floating**
access:	**write only**
mechanism:	**by value**

Sine of the angle specified by **x**. MTH$SIN returns an F_floating number. MTH$DSIN returns a D_floating number. MTH$GSIN returns a G_floating number. Unlike the other three routines, MTH$HSIN returns the sine by reference in the **h_sine** argument.

ARGUMENTS

x

VMS Usage:	**floating_point**
type:	**F_floating, D_floating, G_floating, H_floating**
access:	**read only**
mechanism:	**by reference**

Angle (in radians). The **x** argument is the address of a floating-point number that is this angle. For MTH$SIN, **x** specifies an F_floating number. For MTH$DSIN, **x** specifies a D_floating number. For MTH$GSIN, **x** specifies a G_floating number. For MTH$HSIN, **x** specifies an H_floating number.

h_sine

VMS Usage: **floating_point**
type: **H_floating**
access: **write only**
mechanism: **by reference**

The sine of the angle specified by x. The **h_sine** argument is the address of an H_floating number that is this sine. MTH$HSIN writes the address of the sine into **h_sine**. The **h_sine** argument is used only by the MTH$HSIN routine.

DESCRIPTION

See the MTH$SINCOS routine for the algorithm which is used to compute this sine.

CONDITION VALUE SIGNALED

SS$_ROPRAND	Reserved operand. The MTH$xSIN procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.

MTH$xSINCOS—Sine and Cosine of Angle Expressed in Radians

MTH$xSINCOS returns the sine and the cosine of a given angle (in radians).

FORMAT

MTH$SINCOS *x ,sine ,cosine*
MTH$DSINCOS *x ,sine ,cosine*
MTH$GSINCOS *x ,sine ,cosine*
MTH$HSINCOS *x ,sine ,cosine*

Each of the above four formats accepts as input one of the four floating-point types.

jsb entries

MTH$SINCOS_R5
MTH$DSINCOS_R7
MTH$GSINCOS_R7
MTH$HSINCOS_R7

Each of the above four JSB entries accepts as input one of the four floating-point types.

RETURNS

MTH$SINCOS, MTH$DSINCOS, MTH$GSINCOS, and MTH$HSINCOS return the sine and cosine of the input angle by reference in the **sine** and **cosine** arguments.

ARGUMENTS

x

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating, H_floating**
access: **read only**
mechanism: **by reference**

Angle (in radians) whose sine and cosine are to be returned. The **x** argument is the address of a floating-point number that is this angle. For MTH$SINCOS, **x** is an F_floating number. For MTH$DSINCOS, **x** is a D_floating number. For MTH$GSINCOS, **x** is a G_floating number. For MTH$HSINCOS, **x** is an H_floating number.

sine

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating, H_floating**
access: **write only**
mechanism: **by reference**

Sine of the angle specified by **x**. The **sine** argument is the address of a floating-point number. MTH$SINCOS writes an F_floating number into **sine**. MTH$DSINCOS writes a D_floating number into **sine**. MTH$GSINCOS writes a G_floating number into **sine**. MTH$HSINCOS writes an H_floating number into **sine**.

cosine

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating, H_floating**
access: **write only**
mechanism: **by reference**

Cosine of the angle specified by **x**. The **cosine** argument is the address of a floating-point number. MTH$SINCOS writes an F_floating number into **cosine**. MTH$DSINCOS writes a D_floating number into **cosine**. MTH$GSINCOS writes a G_floating number into **cosine**. MTH$HSINCOS writes an H_floating number into **cosine**.

DESCRIPTION

All routines with JSB entry points accept a single argument in R0:Rm, where *m*, which is defined below, is dependent on the data type.

Data Type	m
F_floating	0
D_floating	1
G_floating	1
H_floating	3

In general, Run-Time Library routines with JSB entry points return one value in R0:Rm. The MTH$SINCOS routine returns two values, however. The sine of **x** is returned in R0:Rm and the cosine of **x** is returned in (R <m+1> :R <2*m+1>).

In radians, the computation of zSIN(X) and zCOS(X) is based on the following polynomial expansions:

$$\text{SIN}(X) = X - X^3/(3!) + X^5/(5!) - X^7/(7!) \ldots$$
$$= X + X*P(X^2),\ \text{where } P(y) = y/(3!) + y^2/(5!) + y^3/(7!) \ldots$$
$$\text{COS}(X) = 1 - X^2/(2!) + x^4/(4!) - X^6/(6!) \ldots$$
$$= Q(X^2),\ \text{where } Q(y) = (1 - y/(2!) + y^2/(4!) + y^3/(6!) \ldots)$$

(1) If $|X| < 2^{(-f/2)}$,
then zSIN(X) = X and zCOS(X) = 1
(see the section on MTH$zCOSH for the definition of f)

(2) If $2^{(-f/2)}$ =< |X| < PI/4,
then $zSIN(X) = X + P(X^2)$
and $zCOS(X) = Q(X^2)$

(3) If PI/4 =< |X| and X > 0,

(a) Let J = INT(X/(PI/4)) and I = J modulo 8

(b) If J is even, let Y = X - J* (PI/4)
otherwise, let Y = (J+1)* (PI/4) - X

With the above definitions, the following table relates zSIN(X) and zCOS(X) to zSIN(Y) and zCOS(Y):

Value of *I*	zSIN(X)	zCOS(X)
0	zSIN(Y)	zCOS(Y)
1	zCOS(Y)	zSIN(Y)
2	zCOS(Y)	-zSIN(Y)
3	zSIN(Y)	-zCOS(Y)
4	-zSIN(Y)	-zCOS(Y)
5	-zCOS(Y)	-zSIN(Y)
6	-zCOS(Y)	zSIN(Y)
7	-zSIN(Y)	zCOS(Y)

```
    (c)   zSIN(Y) and zCOS(Y) are computed as follows:
          zSIN(Y) = Y + P(Y^2),
          and zCOS(Y) = Q(Y^2)
(4)  If PI/4 =< |X| and X < 0,
       then zSIN(X) = -zSIN(|X|)
       and zCOS(X) = zCOS(|X|)
```

CONDITION VALUE RETURNED

SS$_ROPRAND	Reserved operand. The MTH$xSINCOS procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.

MTH$xSINCOSD—Sine and Cosine of Angle Expressed in Degrees

MTH$xSINCOSD returns the sine and cosine of a given angle (in degrees).

FORMAT

MTH$SINCOSD *x ,sine ,cosine*
MTH$DSINCOSD *x ,sine ,cosine*
MTH$GSINCOSD *x ,sine ,cosine*
MTH$HSINCOSD *x ,sine ,cosine*

Each of the above four formats accepts as input one of the four floating-point types.

jsb entries

MTH$SINCOSD_R5
MTH$DSINCOSD_R7
MTH$GSINCOSD_R7
MTH$HSINCOSD_R7

Each of the above four JSB entries accepts as input one of the four floating-point types.

RETURNS

MTH$SINCOSD, MTH$DSINCOSD, MTH$GSINCOSD, and MTH$HSINCOSD return the sine and cosine of the input angle by reference in the **sine** and **cosine** arguments.

ARGUMENTS

x

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating, H_floating**
access: **read only**
mechanism: **by reference**

Angle (in degrees) whose sine and cosine are returned by MTH$xSINCOSD. The **x** argument is the address of a floating-point number that is this angle. For MTH$SINCOSD, **x** is an F_floating number. For MTH$DSINCOSD, **x** is a D_floating number. For MTH$GSINCOSD, **x** is a G_floating number. For MTH$HSINCOSD, **x** is an H_floating number.

sine

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating, H_floating**
access: **write only**
mechanism: **by reference**

Sine of the angle specified by **x**. The **sine** argument is the address of a floating-point number. MTH$SINCOSD writes an F_floating number into **sine**. MTH$DSINCOSD writes a D_floating number into **sine**.

MTH$GSINCOSD writes a G_floating number into **sine**. MTH$HSINCOSD writes an H_floating number into **sine**.

cosine

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating, H_floating**
access: **write only**
mechanism: **by reference**

Cosine of the angle specified by **x**. The **cosine** argument is the address of a floating-point number. MTH$SINCOSD writes an F_floating number into **cosine**. MTH$DSINCOSD writes a D_floating number into **cosine**. MTH$GSINCOSD writes a G_floating number into **cosine**. MTH$HSINCOSD writes an H_floating number into **cosine**.

DESCRIPTION

All routines with JSB entry points accept a single argument in R0:Rm, where *m*, which is defined below, is dependent on the data type.

Data Type	m
F_floating	0
D_floating	1
G_floating	1
H_floating	3

In general, Run-Time Library routines with JSB entry points return one value in R0:Rm. The MTH$SINCOSD routine returns two values, however. The sine of **x** is returned in R0:Rm and the cosine of **x** is returned in (R <m+1> :R <2•m+1>).

In degrees, the computation of zSIND(X) and zCOSD(X) is based on the following polynomial expansions:

```
SIND(X) = (C*X) - (C*X)^3/(3!) + (C*X)^5/(5!) -  (C*X)^7/(7!) ...
        = X/2^6 + X*P(X^2),
          where P(y) = -y/(3!) + y^2/(5!) - y^3/(7!) ...
COSD(X) = 1 - (C*X)^2/(2!) + (C*X)^4/(4!) -(C*X)^6/(6!) ...
        = Q(X^2), where Q(y) = 1 - y/(2!) + y^2/(4!) - y^3/(6!) ...
          and C = PI/180
```

1 If $|X| < (180/PI)\bullet 2^{-2^{(e-1)}}$ and underflow signaling is enabled, underflow is signaled for zSIND(X) and zSINCOSD(X). See MTH$zCOSH for the definition of *e*.

OTHERWISE:

2 If $|X| < (180/PI)\bullet 2^{(-f/2)}$, then $zSIND(X) = (PI/180)\bullet X$ and $zCOSD(X) = 1$. See MTH$zCOSH for the definition of *f*.

3 If $(180/PI)\bullet 2^{(-f/2)} =< |X| < 45$ then $zSIND(X) = X/2^6 + P(X^2)$ and $zCOSD(X) = Q(X^2)$

4 If $45 =< |X|$ *and* $X > 0$,

a Let $J = INT(X/(45))$ and $I = J$ *modulo 8*

b If J is even, let $Y = X - J{\bullet}45$; otherwise, let $Y = (J+1){\bullet}45 - X$. With the above definitions, the following table relates zSIND(X) and zCOSD(X) to zSIND(Y) and zCOSD(Y):

Value of *I*	zSIND(X)	zCOSD(X)
0	zSIND(Y)	zCOSD(Y)
1	zCOSD(Y)	zSIND(Y)
2	zCOSD(Y)	-zSIND(Y)
3	zSIND(Y)	-zCOSD(Y)
4	-zSIND(Y)	-zCOSD(Y)
5	-zCOSD(Y)	-zSIND(Y)
6	-zCOSD(Y)	zSIND(Y)
7	-zSIND(Y)	zCOSD(Y)

c zSIND(Y) and zCOSD(Y) are computed as follows:

```
zSIND(Y) = Y/2^6 + P(Y^2)
zCOSD(Y) = Q(Y^2)
```

d If $45 =< |X|$ *and* $X < 0$, then *zSIND(X) = -zSIND(|X|)* and *zCOSD(X) = zCOSD(|X|)*

CONDITION VALUES SIGNALED

SS$_ROPRAND	Reserved operand. The MTH$xSINCOSD procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.
MTH$_FLOUNDMAT	Floating-point underflow in Math Library. The absolute value of the input angle is less than 180/PI•2••-m (where m = 128 for F_floating and D_floating, 1,024 for G_floating, and 16,384 for H_floating).

MTH$xSIND—Sine of Angle Expressed in Degrees

MTH$xSIND returns the sine of a given angle (in degrees).

FORMAT

MTH$SIND *x*
MTH$DSIND *x*
MTH$GSIND *x*
MTH$HSIND *h_sine ,x*

Each of the above four formats accepts as input one of the four floating-point types.

jsb entries

MTH$SIND_R4
MTH$DSIND_R7
MTH$GSIND_R7
MTH$HSIND_R5

Each of the above four JSB entries accepts as input one of the four floating-point types.

RETURNS

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating**
access: **write only**
mechanism: **by value**

The sine of the angle. MTH$SIND returns an F_floating number. MTH$DSIND returns a D_floating number. MTH$GSIND returns a G_floating number. Unlike the other three routines, MTH$HSIND returns the angle by reference in the **h_sine** argument.

ARGUMENTS

x

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating, H_floating**
access: **read only**
mechanism: **by reference**

Angle (in degrees). The **x** argument is the address of a floating-point number that is this angle. For MTH$SIND, **x** specifies an F_floating number. For MTH$DSIND, **x** specifies a D_floating number. For MTH$GSIND, **x** specifies a G_floating number. For MTH$HSIND, **x** specifies an H_floating number.

h_sine

VMS Usage: **floating_point**
type: **H_floating**
access: **write only**
mechanism: **by reference**

Sine of the angle specified by **x**. The **h_sine** argument is the address of an H_floating number that is this sine. MTH$HSIND writes the address of the angle into **h_sine**. The **h_sine** argument is used only by the MTH$HSIND routine.

DESCRIPTION

See MTH$SINCOSD for the algorithm that is used to compute the sine.

CONDITION VALUES SIGNALED

SS$_ROPRAND	Reserved operand. The MTH$SIND procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased ecponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.
MTH$_FLOUNDMAT	Floating-point underflow in Math Library. The absolute value of the input angle is less than *180/PI*2**-m* (where m = 128 for F_floating and D_floating, 1,024 for G_floating, and 16,384 for H_floating).

MTH$xSINH—Hyperbolic Sine

MTH$xSINH returns the hyperbolic sine of the input value specified by **x**.

FORMAT

MTH$SINH *x*
MTH$DSINH *x*
MTH$GSINH *x*
MTH$HSINH *h_sinh ,x*

Each of the above four formats accepts as input one of the four floating-point types.

RETURNS

VMS Usage:	**floating_point**
type:	**F_floating, D_floating, G_floating**
access:	**write only**
mechanism:	**by value**

The hyperbolic sine of **x**. MTH$SINH returns an F_floating number. MTH$DSINH returns a D_floating number. MTH$GSINH returns a G_floating number. However, unlike the other three routines, MTH$HSINH returns the hyperbolic sine by reference in the **h_sinh** argument.

ARGUMENTS

x

VMS Usage:	**floating_point**
type:	**F_floating, D_floating, G_floating, H_floating**
access:	**read only**
mechanism:	**by reference**

The input value. The **x** argument is the address of a floating-point number that is this value. For MTH$SINH, **x** specifies an F_floating number. For MTH$DSINH, **x** specifies a D_floating number. For MTH$GSINH, **x** specifies a G_floating number. For MTH$HSINH, **x** specifies an H_floating number.

h_sinh

VMS Usage:	**floating_point**
type:	**H_floating**
access:	**write only**
mechanism:	**by reference**

Hyperbolic sine of the input value specified by **x**. The **h_sinh** argument is the address of an H_floating number that is this hyperbolic sine. MTH$HSINH writes the address of the hyperbolic sine into **h_sinh**. The **h_sinh** argument is used only by the MTH$HSINH routine.

DESCRIPTION

Computation of the hyperbolic sine function depends on the magnitude of the input argument. The range of the function is partitioned using four data type dependent constants: a(z), b(z), c(z), and d(z). The subscript *z* indicates the data type. The constants depend on the number of exponent bits (*e*) and the number of fraction bits (*f*) associated with the data type (*z*).

The values of *e* and *f* are:

z	e	f
F	8	24
D	8	56
G	11	53
H	15	113

The values of the constants in terms of *e* and *f* are:

Variable	Value
a(z)	$2^{(-f/2)}$
b(z)	CEILING[$(f+1)/2*ln(2)$] for F,D, and G
	$(f+1)/2*ln(2)$ for H
c(z)	$(2^{(e-1)-1})*ln(2)$
d(z)	$c(z)+ln(2)$

Based on the above definitions, zSINH(X) is computed as follows:

Value of x	Value Returned
$\|X\| < a(z)$	X
$a(z) =< \|X\| < .25$	zSINH(X) is computed using a power series expansion in $\|X\|^2$
$.25 =< \|X\| < b(z)$	(zEXP(X) - zEXP(-X))/2
$b(z) =< \|X\| < c(z)$	SIGN(X)*zEXP(|X|)/2
$c(z) =< \|X\| < d(z)$	SIGN(X)*zEXP(|X|-ln(2))
$d(z) =< \|x\|$	Overflow occurs

CONDITION VALUES SIGNALED

SS$_ROPRAND

Reserved operand. The MTH$xSINH procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.

MTH$_FLOOVEMAT

Floating-point overflow in Math Library: the absolute value of **x** is greater than *yyy*. LIB$SIGNAL copies the floating-point reserved operand to the mechanism argument vector CHF$L_MCH_SAVR0/R1. The result is the floating-point reserved operand unless you have written a condition handler to change CHF$L_MCH_SAVR0/R1. The values of *yyy* are approximately:

MTH$SINH	88.722
MTH$DSINH	88.722
MTH$GSINH	709.782
MTH$HSINH	11356.523

MTH$xSQRT—Square Root

MTH$xSQRT returns the square root of the input value **x**.

FORMAT

MTH$SQRT *x*
MTH$DSQRT *x*
MTH$GSQRT *x*
MTH$HSQRT *h_sqrt ,x*

Each of the above four formats accepts as input one of the four floating-point types.

jsb entries

MTH$SQRT_R3
MTH$DSQRT_R5
MTH$GSQRT_R5
MTH$HSQRT_R8

Each of the above four JSB entries accepts as input one of the four floating-point types.

RETURNS

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating**
access: **write only**
mechanism: **by value**

The square root of **x**. MTH$SQRT returns an F_floating number. MTH$DSQRT returns a D_floating number. MTH$GSQRT returns a G_floating number. Unlike the other three routines, MTH$HSQRT returns the square root in the **h_sqrt** argument.

ARGUMENTS

x

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating, H_floating**
access: **read only**
mechanism: **by reference**

Input value. The **x** argument is the address of a floating-point number that contains this input value. For MTH$SQRT, **x** specifies an F_floating number. For MTH$DSQRT, **x** specifies a D_floating number. For MTH$GSQRT, **x** specifies a G_floating number. For MTH$HSQRT, an H_floating number.

h_sqrt

VMS Usage: **floating_point**
type: **H_floating**
access: **write only**
mechanism: **by reference**

Square root of the input value specified by **x**. The **h_sqrt** argument is the address of an H_floating number that is this square root. MTH$HSQRT

writes the address of the square root into **h_sqrt**. The **h_sqrt** argument is used only by the MTH$HSQRT routine.

DESCRIPTION

The square root of X is computed as follows:

If X < 0, an error is signaled.

Let X = 2^{K} * F

where:
K is the exponential part of the floating-point data
F is the fractional part of the floating-point data

If K is even:
  X = $2^{(2*P)}$ * F,
  zSQRT(X) = 2^{P} * zSQRT(F),
  1/2 =< F < 1, where P = K/2

If K is odd:
  X = $2^{(2*P+1)}$ * F = $2^{(2*P+2)}$ * (F/2),
  zSQRT(X) = $2^{(P+1)}$ * zSQRT(F/2),
  1/4 =< F/2 < 1/2, where p = (K-1)/2

Let F' = A*F + B, when K is even:

A = 0.453730314 (octal)
B = 0.327226214 (octal)

Let F' = A* + B, when K is odd:

A = 0.650117146 (octal)
B = 0.230170444 (octal)

Let K' = P, when K is even
Let K' = P+1, when K is odd

Let Y[0] = $2^{K\oplus}$ * F' be a straight line approximation within the given interval using coefficients A and B which minimize the absolute error at the midpoint and endpoint.

Starting with Y[0], n Newton-Raphson iterations are performed:

Y[n+1] = 1/2 * (Y[n] + X/Y[n])

where n = 2, 3, 3, or 5 for z = F_floating, D_floating, G_floating, or H_floating respectively.

CONDITION VALUES SIGNALED

SS$_ROPRAND	Reserved operand. The MTH$xSQRT procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.
MTH$_SQUROONEG	Square root of negative number. Argument **x** is less than 0.0. LIB$SIGNAL copies the floating-point reserved operand to the mechanism argument vector CHF$L_MCH_SAVR0/R1. The result is the floating-point reserved operand unless you have written a condition handler to change CHF$L_MCH_SAVR0/R1.

MTH$xTAN—Tangent of Angle Expressed in Radians

MTH$xTAN returns the tangent of a given angle (in radians).

FORMAT

MTH$TAN *x*
MTH$DTAN *x*
MTH$GTAN *x*
MTH$HTAN *h_tan ,x*

Each of the above four formats accepts as input one of the four floating-point types.

jsb entries

MTH$TAN_R4
MTH$DTAN_R7
MTH$GTAN_R7
MTH$HTAN_R5

Each of the above four JSB entries accepts as input one of the four floating-point types.

RETURNS

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating**
access: **write only**
mechanism: **by reference**

The tangent of the angle specified by **x**. MTH$TAN returns an F_floating number. MTH$DTAN returns a D_floating number. MTH$GTAN returns a G_floating number. Unlike the other three routines, MTH$HTAN returns an H_floating number by reference in the **h_tan** argument.

ARGUMENTS

x

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating, H_floating**
access: **read only**
mechanism: **by value**

The input angle (in radians). The **x** argument is the address of a floating-point number that is this angle. For MTH$TAN, **x** specifies an F_floating number. For MTH$DTAN, **x** specifies a D_floating number. For MTH$GTAN, **x** specifies a G_floating number. For MTH$HTAN, **x** specifies an H_floating number.

h_tan

VMS Usage: **floating_point**
type: **H_floating**
access: **write only**
mechanism: **by reference**

Tangent of the angle specified by x. The **h_tan** argument is the address of an H_floating number that is this tangent. MTH$HTAN writes the address of the tangent into **h_tan**. The **h_tan** argument is used only by the MTH$HTAN routine.

DESCRIPTION

When the input argument is expressed in radians, the tangent function is computed as follows:

1. If $|X| < 2^{(-f/2)}$, then $zTAN(X) = X$ (see the section on MTH$zCOSH for the definition of f)
2. Otherwise, call MTH$zSINCOS to obtain zSIN(X) and zCOS(X); then
   a. If $zCOS(X) = 0$, signal overflow
   b. Otherwise, $zTAN(X) = zSIN(X)/zCOS(X)$

CONDITION VALUES SIGNALED

SS$_ROPRAND	Reserved operand. The MTH$xTAN procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.
MTH$_FLOOVEMAT	Floating-point overflow in Math Library.

MTH$xTAND—Return Tangent of Angle Expressed in Degrees

MTH$xTAND returns the tangent of a given angle (in degrees).

FORMAT

MTH$TAND *x*
MTH$DTAND *x*
MTH$GTAND *x*
MTH$HTAND *h_tan ,x*

Each of the above four formats accepts as input one of the four floating-point types.

jsb entries

MTH$TAND_R4
MTH$DTAND_R7
MTH$GTAND_R7
MTH$HTAND_R5

Each of the above four JSB entries accepts as input one of the four floating-point types.

RETURNS

VMS Usage:	**floating_point**
type:	**F_floating, D_floating, G_floating**
access:	**write only**
mechanism:	**by value**

Tangent of the angle specified by **x**. MTH$TAND returns an F_floating number. MTH$DTAND returns a D_floating number. MTH$GTAND returns a G_floating number. Unlike the other three routines, MTH$HTAND returns the angle in the **h_tan** argument.

ARGUMENTS

x

VMS Usage:	**floating_point**
type:	**F_floating, D_floating, G_floating, H_floating**
access:	**read only**
mechanism:	**by reference**

The input angle (in degrees). The **x** argument is the address of a floating-point number which is this angle. For MTH$TAND, **x** specifies an F_floating number. For MTH$DTAND, **x** specifies a D_floating number. For MTH$GTAND, **x** specifies a G_floating number. For MTH$HTAND, **x** specifies an H_floating number.

h_tan

VMS Usage: **floating_point**
type: **H_floating**
access: **write only**
mechanism: **by reference**

Tangent of the angle specified by **x**. The **h_tan** argument is the address of an H_floating number that is this tangent. MTH$HTAND writes the address of the tangent into **h_tan**. The **h_tan** argument is used only by the MTH$HTAN routine.

DESCRIPTION

When the input argument is expressed in degrees, the tangent function is computed as follows:

1. If $|X| < (180/PI)*2^{(-2)}$ and underflow signaling is enabled, underflow is signaled (see the section on MTH$zCOSH for the definition of *e*).
2. Otherwise, if $|X| < (180/PI)*2^{(-f/2)}$, then $zTAND(X) = (PI/180)*X$. See the description of MTH$zCOSH for the definition of *f*.
3. Otherwise, call MTH$zSINCOSD to obtain zSIND(X) and zCOSD(X).
   a. Then, if $zCOSD(X) = 0$, signal overflow
   b. Else, $zTAND(X) = zSIND(X)/zCOSD(X)$

CONDITION VALUES SIGNALED

SS$_ROPRAND	Reserved operand. The MTH$xTAND procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.
MTH$_FLOOVEMAT	Floating-point overflow in Math Library.

MTH$xTANH—Compute the Hyperbolic Tangent

MTH$xTANH returns the hyperbolic tangent of the input value.

FORMAT

MTH$TANH *x*
MTH$DTANH *x*
MTH$GTANH *x*
MTH$HTANH *h_tanh ,x*

Each of the above four formats accepts as input one of the four floating-point types.

RETURNS

VMS Usage:	**floating_point**
type:	**F_floating, D_floating, G_floating**
access:	**write only**
mechanism:	**by value**

The hyperbolic tangent of **x**. MTH$TANH returns an F_floating number. MTH$DTANH returns a D_floating number. MTH$GTANH returns a G_floating number. Unlike the other three routines, MTH$HTANH returns the hyperbolic tangent by reference in the **h_tanh** argument.

ARGUMENTS

x

VMS Usage:	**floating_point**
type:	**F_floating, D_floating, G_floating, H_floating**
access:	**read only**
mechanism:	**by reference**

The input value. The **x** argument is the address of a floating-point number that contains this input value. For MTH$TANH, **x** specifies an F_floating number. For MTH$DTANH, **x** specifies a D_floating number. For MTH$GTANH, **x** specifies a G_floating number. For MTH$HTANH, **x** specifies an H_floating number.

h_tanh

VMS Usage:	**floating_point**
type:	**H_floating**
access:	**write only**
mechanism:	**by reference**

Hyperbolic tangent of the value specified by **x**. The **h_tanh** argument is the address of a H_floating number that is this hyperbolic tangent. MTH$HTANH writes the address of the hyperbolic tangent into **h_tanh**. The **h_tanh** argument is used only by the MTH$TANH routine.

DESCRIPTION

For MTH$HTANH, the hyperbolic tangent of x is computed as follows:

Value of x	Hyperbolic Tangent Returned
$\lvert X\rvert =< 2^{-g}$	X
$2^{-g} < \lvert X\rvert =< 0.25$	zSINH(X) / zCOSH(X)
$0.25 < \lvert X\rvert < h$	(zEXP(2*X) - 1) / (zEXP(2*X) + 1)
$h =< \lvert X\rvert$	sign(X) * 1

where g = 12, 28, 26, or 56 and h = 10, 21, 20, or 40 for z = F_floating, D_floating, G_floating, or H_floating, respectively.

For MTH$TANH, MTH$DTANH, and MTH$GTANH the hyperbolic tangent of **x** is computed as follows:

Value of x	Hyperbolic Tangent Returned
|x| =< 2**-g	X
2**-g < |X| =< 0.5	xTANH(X) = X + X^3 * R(X^2), where R(X^2) is a rational function of X^2.
0.5 < |X| < 1.0	xTANH(X) = xTANH(xHI) + xTANH(xLO)*C/B
	where C = 1 - xTANH(xHI)*xTANH(xHI), B = 1 + xTANH(xHI)*xTANH(xLO), xHI = 1/2 + N/16 + 1/32 for N=0,1,...,7, and xLO = X - xHI.
1.0 < |X| < h	xTANH(X) = (xEXP(2*X) - 1) / (xEXP(2*X) + 1)
h =< |X|	xTANH(X) = sign(X) * 1

where g is as above and h = 10, 20, 19 for F, D, G.

CONDITION VALUE SIGNALED

SS$_ROPRAND	Reserved operand. The MTH$xTANH procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.

MTH$UMAX—Compute Unsigned Maximum

MTH$UMAX computes the unsigned longword maximum of n unsigned longword arguments, where n is greater than or equal to 1.

FORMAT

MTH$UMAX *arg1 [... ,argn]*

RETURNS

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

Maximum value returned by MTH$UMAX.

ARGUMENTS

arg1

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

First of the arguments of which MTH$UMAX computes the maximum. The **arg1** argument is an unsigned longword which contains the first value.

argn

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Last of the arguments of which MTH$UMAX computes the maximum. The **argn** argument is an unsigned longword which contains the last of the values which MTH$UMAX compares to find the maximum.

DESCRIPTION

MTH$UMIN is the unsigned version of MTH$JMAX0.

CONDITION VALUES RETURNED

None.

MTH$UMIN—Compute Unsigned Minimum

MTH$UMIN computes the unsigned longword minimum of n unsigned longword arguments, where n is greater than or equal to 1.

FORMAT

MTH$UMIN *arg1 [... ,argn]*

RETURNS

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Minimum value returned by MTH$UMIN.

ARGUMENTS

arg1

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

First of the arguments of which MTH$UMIN computes the minimum. The **arg1** argument is an unsigned longword which contains the first value.

argn

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Last of the arguments of which MTH$UMIN computes the minimum. The **argn** argument is an unsigned longword which contains the last of the values which MTH$UMIN compares to find the minimum.

DESCRIPTION

MTH$UMIN is the unsigned version of MTH$JMIN0.

SHORT

None.

OTS$CVT_L_TB—Convert an Unsigned Integer to Binary Text

OTS$CVT_L_TB converts an unsigned integer value of arbitrary length to binary representation in an ASCII text string. By default, a longword is converted.

FORMAT

OTS$CVT_L_TB *value ,out-str [,int-digits] [,value-size]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

value

VMS Usage: **varying_arg**
type: **unspecified**
access: **read only**
mechanism: **by reference**

Unsigned integer value that OTS$CVT_L_TB converts to binary representation in an ASCII text string. The **value** argument is the address of this integer value.

out-str

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor, fixed-length**

ASCII text string that OTS$CVT_L_TB creates when it converts the integer value. The **out-str** argument is the address of a descriptor pointing to this ASCII text string. The string is assumed to be fixed length (DSC$K_CLASS_S).

int-digits

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by value**

Minimum number of digits in the binary representation to be generated. The **int-digits** argument is a signed longword integer containing this minumum number. This is an optional argument. If omitted, the default is 1. If the actual number of significant digits is less than the minimum number of digits, leading zeros are produced. If the minimum number of digits is zero and the value of the integer to be converted is also zero, OTS$CVT_L_TB creates a blank string.

value-size

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by value**

Size of the integer to be converted, in bytes. The **value-size** argument is a signed longword integer containing the byte size. This is an optional argument. If omitted, the default is 4.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
OTS$_OUTCONERR	Output conversion error. The result would have exceeded the fixed-length string; the output string is filled with asterisks.

EXAMPLE

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
     FTTY     D   F       4            TTY
     C* Initialize numeric value to be converted.
     C                      Z-ADD13        VALUE   90
     C            CVTLTB    EXTRN'OTS$CVT_L_TB'
     C* Convert the number to binary in a string.
     C                      CALL CVTLTB
     C                      PARM           VALUE      RL
     C                      PARMD          OUTSTR  4
     C* Display on the terminal the converted string.
     C            OUTSTR    DSPLYTTY
     C                      SETON                     LR
```

The RPG II program above displays the string '1101' on the terminal.

OTS$CVT_L_TI—Convert Signed Integer to Decimal Text

OTS$CVT_L_TI converts a signed integer to a decimal ASCII text string. This procedure supports FORTRAN Iw and Iw.m output and BASIC output conversion.

FORMAT

OTS$CVT_L_TI *value ,out-str [,int-digits] [,value-size] [,flags]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

value

VMS Usage: **varying_arg**
type: **unspecified**
access: **read only**
mechanism: **by reference**

Signed integer value that OTS$CVT_L_TI converts to a decimal ASCII text string. The **value** argument is the address of this integer value.

out-str

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor, fixed-length**

Decimal ASCII text string that OTS$CVT_L_TI creates when it converts the signed integer. The **out-str** argument is the address of a descriptor pointing to this text string. The string is assumed to be fixed length (DSC$K_CLASS_S).

int-digits

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by value**

Minimum number of digits to be generated when OTS$CVT_L_TI converts the signed integer to a decimal ASCII text string. The **int-digits** argument is a signed longword integer containing this number. This is an optional argument. If omitted, the default value is 1. If the actual number of significant digits is smaller, OTS$CVT_L_TI inserts leading zeros into the output string. If **int-digits** is zero and **value** is zero, OTS$CVT_L_TI writes a blank string to the output string.

value-size

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by value**

Number of bytes occupied by the value to be converted to text. The **value-size** argument is a signed longword integer containing this value size. The value size must be either 1, 2, or 4. If value size is 1 or 2, the value is sign-extended to a longword before conversion. This is an optional argument. If omitted, the default is 4.

flags

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Caller-supplied flags that you may use if you want OTS$CVT_L_TI to insert a plus sign before the converted number. The **flags** argument is an unsigned longword containing the flags.

The caller flags are defined as follows:

Bit 0	If set, a plus sign (+) will be inserted before the first nonblank character in the output string; otherwise, the plus sign will be omitted.

This is an optional argument. If **flags** is omitted, all bits are clear and the plus sign is not inserted.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
OTS$_OUTCONERR	Output conversion error. The result would have exceeded the fixed-length string; the output string is filled with asterisks.

OTS$CVT_L_TL—Convert Integer to Logical Text

OTS$CVT_L_TL converts an integer to ASCII text string representation using FORTRAN L (logical) format.

FORMAT

OTS$CVT_L_TL *value ,out-str*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

value

VMS Usage:	**longword_signed**
type:	**longword integer (signed)**
access:	**read only**
mechanism:	**by reference**

Value that OTS$CVT_L_TL converts to an ASCII text string. The **value** argument is the address of a signed longword integer containing this integer value.

out-str

VMS Usage:	**char_string**
type:	**character string**
access:	**write only**
mechanism:	**by descriptor, fixed-length**

Output string that OTS$CVT_L_TL creates when it converts the integer value to an ASCII text string. The **out-str** argument is the address of a descriptor pointing to this ASCII text string.

The output string is assumed to be fixed length (DSC$K_CLASS_S).

The output string consists of *(length -1)* blanks followed by the letter T if bit 0 is set, or the letter F if bit 0 is clear.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
OTS$_OUTCONERR	Output conversion error. The result would have exceeded the fixed-length string; the output string is of zero length (DSC$W_LENGTH=0).

EXAMPLE

```
5 !+
  ! This is an example program
  ! showing the use of OTS$CVT_L_TL.
  !-
  VALUE% = 10
  OUTSTR$ = ' '
  CALL OTS$CVT_L_TL(VALUE%, OUTSTR$)
  PRINT OUTSTR$
9 END
```

This BASIC example illustrates the use of OTS$CVT_L_TL. The output generated by this program is "F".

OTS$CVT_L_TO—Convert Unsigned Integer to Octal Text

OTS$CVT_L_TO converts an unsigned integer to an octal ASCII text string. OTS$CVT_L_TO supports FORTRAN Ow and Ow.m output conversion formats.

FORMAT

OTS$CVT_L_TO *value ,out-str [,int-digits] [,value-size]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

value

VMS Usage: **varying_arg**
type: **unspecified**
access: **read only**
mechanism: **by reference**

Integer value that OTS$CVT_L_TO converts to an octal ASCII text string. The **value** argument is the address of this integer value.

out-str

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor, fixed-length**

Output string that OTS$CVT_L_TO creates when it converts the integer value to an octal ASCII text string. The **out-str** argument is the address of a descriptor pointing to the octal ASCII text string. The string is assumed to be fixed length (DSC$K_CLASS_S).

int-digits

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by value**

Minimum number of digits that OTS$CVT_L_TO generates when it converts the integer value to an octal ASCII text string. The **int-digits** argument is a signed longword integer containing the minimum number of digits. This is an optional argument. If omitted, the default is 1. If the actual number of significant digits in the octal ASCII text string is less than the minimum number of digits, OTS$CVT_L_TO inserts leading zeros into the output string. If **int-digits** is zero and **value** is zero, OTS$CVT_L_TO writes a blank string to the output string.

value-size

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by value**

Size of the integer to be converted, in bytes. The **value-size** argument is a signed longword integer containing the number of bytes in the integer to be converted by OTS$CVT_L_TO. This is an optional argument. If omitted, the default is 4.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
OTS$_OUTCONERR	Output conversion error. The result would have exceeded the fixed-length string; the output string is filled with asterisks.

OTS$CVT_L_TU—Convert Unsigned Integer to Decimal Text

OTS$CVT_L_TU converts a byte, word or longword value to unsigned decimal representation in an ASCII text string. By default, a longword is converted.

FORMAT

OTS$CVT_L_TU *value ,out-str [,int-digits] [,value-size]*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

value

VMS Usage:	**varying_arg**
type:	**unspecified**
access:	**read only**
mechanism:	**by reference**

Unsigned byte, word or longword that OTS$CVT_L_TU converts to unsigned decimal representation in an ASCII text string. (The value of the **value-size** argument determines whether **value** is a byte, word or longword.) The **value** argument is the address of the unsigned integer.

out-str

VMS Usage:	**char_string**
type:	**character string**
access:	**write only**
mechanism:	**by descriptor, fixed-length**

Output string (fixed-length) that OTS$CVT_L_TU creates when it converts the integer value to unsigned decimal representation in an ASCII text string. The **out-str** argument is the address of a descriptor pointing to this ASCII text string.

int-digits

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by value**

Minimum number of digits in the ASCII text string that OTS$CVT_L_TU creates. The **int-digits** argument is an unsigned longword containing the minimum number. This is an optional argument. If omitted, the default is 1.

If the actual number of significant digits in the output string created is less than the minimum number, OTS$CVT_L_TU inserts leading zeros into the output string. If the minimum number of digits is zero and the integer value to be converted is also zero, OTS$CVT_L_TU writes a blank string to the output string.

value-size

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Size of the integer value to be converted, in bytes. The **value-size** argument is an unsigned longword containing the size of the integer value. This is an optional argument. If omitted, the default is 4. The only values that OTS$CVT_L_TU allows are 1, 2 and 4. If any other value is specified, OTS$CVT_L_TU uses the default value, 4.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
OTS$_OUTCONERR	Output conversion error. The result would have exceeded the fixed-length string; the output string is filled with asterisks.

EXAMPLE

```
         0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     FTTY     D  F       7             TTY
     C* Initialize numeric value to be converted.
     C                   Z-ADD32857     VALUE   90
     C                   Z-ADD7         DIGITS  90
     C         CVTLTU    EXTRN'OTS$CVT_L_TU'
     C* Convert the number to decimal in a string with 7 decimal digits.
     C                   CALL CVTLTU
     C                   PARM           VALUE      RL
     C                   PARMD          OUTSTR  7
     C                   PARMV          DIGITS
     C* Display on the terminal the converted string.
     C         OUTSTR    DSPLYTTY
     C                   SETON                     LR
```

The RPG II program above displays the string '0032857' on the terminal screen.

OTS$CVT_L_TZ—Convert Integer to Hexadecimal Text

OTS$CVT_L_TZ converts an unsigned integer to a hexadecimal ASCII text string. OTS$CVT_L_TZ supports FORTRAN Zw and Zw.m output conversion formats.

FORMAT

OTS$CVT_L_TZ *value ,out-str [,int-digits] [,value-size]*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

value

VMS Usage:	**varying_arg**
type:	**unspecified**
access:	**read only**
mechanism:	**by reference**

Integer value that OTS$CVT_L_TZ converts to a hexadecimal ASCII text string. The **value** argument is the address of this integer value.

out-str

VMS Usage:	**char_string**
type:	**character string**
access:	**write only**
mechanism:	**by descriptor, fixed-length**

Output string that OTS$CVT_L_TZ creates when it converts the integer value to a hexidecimal ASCII text string. The **out-str** argument is the address of a descriptor pointing to this ASCII text string. The string is assumed to be fixed length (DSC$K_CLASS_S).

int-digits

VMS Usage:	**longword_signed**
type:	**longword integer (signed)**
access:	**read only**
mechanism:	**by value**

Minimum number of digits in the ASCII text string that OTS$CVT_L_TZ creates when it converts the integer. The **int-digits** argument is a signed longword integer containing this minimum number. This is an optional argument. If omitted, the default is 1. If the actual number of significant digits in the text string that OTS$CVT_L_TZ creates is less than this minimum number, OTS$CVT_L_TZ inserts leading zeros in the output string. If the minimum number of digits is zero and the integer value to be converted is also zero, OTS$CVT_L_TZ writes a blank string to the output string.

value-size

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by value**

Size of the integer that OTS$CVT_L_TZ converts, in bytes. The **value-size** argument is a signed longword integer containing the value size. This is an optional argument. If omitted, the default is 4.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
OTS$_OUTCONERR	Output conversion error. The result would have exceeded the fixed-length string; the output string is filled with asterisks.

EXAMPLE

```
with TEXT_IO; use TEXT_IO;
procedure SHOW_CONVERT is
   type INPUT_INT is new INTEGER range 0..INTEGER'LAST;
   INTVALUE : INPUT_INT := 256;
   HEXSTRING : STRING(1..11);
   procedure CONVERT_TO_HEX (I : in INPUT_INT; HS : out STRING);
   pragma INTERFACE (RTL, CONVERT_TO_HEX);
   pragma IMPORT_PROCEDURE (INTERNAL => CONVERT_TO_HEX,
                            EXTERNAL => "OTS$CVT_L_TZ",
                            MECHANISM =>(REFERENCE,
                                   DESCRIPTOR (CLASS => S)));
begin
   CONVERT_TO_HEX (INTVALUE, HEXSTRING);
   PUT_LINE("This is the value of HEXSTRING");
   PUT_LINE(HEXSTRING);
end;
```

This Ada example uses OTS$CVT_L_TZ to convert a longword integer to hexadecimal text.

OTS$CNVOUT—Convert D_floating, G_floating or H_floating to Character String

OTS$CNVOUT, OTS$CNVOUT_G and OTS$CNVOUT_H convert a D_floating, G_floating or H_floating number to a character string in the FORTRAN E format.

FORMAT

OTS$CNVOUT *value ,out-string ,digits-in-fract*
OTS$CNVOUT_G *value ,out-string ,digits-in-fract*
OTS$CNVOUT_H *value ,out-string ,digits-in-fract*

RETURNS

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

value

VMS Usage:	**floating_point**
type:	**D_floating, G_floating, H_floating**
access:	**read only**
mechanism:	**by reference**

Value that OTS$CNVOUT converts to a character string. For OTS$CNVOUT, the **value** argument is the address of a D_floating number containing the value. For OTS$CNVOUT_G, the **value** argument is the address of a G_floating number containing the value. For OTS$CNVOUT_H, the **value** argument is the address of an H_floating number containing the value.

out-string

VMS Usage:	**char_string**
type:	**character string**
access:	**write only**
mechanism:	**by descriptor, fixed length**

Output string into which OTS$CNVOUT writes the character string result of the conversion. The **out-string** argument is the address of a descriptor pointing to the output string.

digits-in-fract

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by value**

Number of digits in the fractional portion of the result. The **digits-in-fract** argument is an unsigned longword containing the number of digits to be written to the fractional portion of the result.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
SS$_ROPRAND	Floating reserved operand detected.
OTS$_OUTCONERR	Output conversion error. The result would have exceeded the fixed-length string; the output string is filled with asterisks.

OTS$CVT_TB_L—Convert Binary Text to Unsigned Integer

OTS$CVT_TB_L converts an ASCII text string representation of an unsigned binary value to an unsigned integer value of arbitrary length. By default, the result is a longword. Valid input characters are the blank and the digits 0 and 1. No sign is permitted.

FORMAT

OTS$CVT_TB_L *inp-str ,value [,value-size] [,flags]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

inp-str

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Input string containing the ASCII text string representation of an unsigned binary value that OTS$CVT_TB_L converts to an integer value. The **inp-str** argument is the address of a descriptor pointing to the ASCII text string.

value

VMS Usage: **varying_arg**
type: **unspecified**
access: **write only**
mechanism: **by reference**

Integer that OTS$CVT_TB_L creates when it converts the ASCII text string. The **value** argument is the address of the integer value.

value-size

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by value**

Number of bytes occupied by the value created when OTS$CVT_TB_L converts the ASCII text string to an integer value. The **value-size** argument contains the value size. If **value-size** contains a zero or a negative number, OTS$CVT_TB_L returns an error code as the condition value. This is an optional argument. If omitted, the default is 4.

flags

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

User-supplied flags that OTS$CVT_TB_L uses to determine how to interpret blanks and tabs. The **flags** argument contains the value of the user-supplied flags.

The flags are defined as follows:

Bit	Description
0	If set, OTS$CVT_TB_L ignores blanks. If clear, OTS$CVT_TB_L interprets blanks as zeros.
4	If set, OTS$CVT_TB_L ignores tabs. If clear, OTS$CVT_TB_L interprets tabs as invalid characters.

This is an optional argument. The default is that all bits are clear.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
OTS$_INPCONERR	Input conversion error. An invalid character, overflow, or invalid **value-size** occurred.

EXAMPLE

```
1
   OPTION                           &
       TYPE = EXPLICIT

   !+
   !   This program demonstates the use of OTS$CVT_TB_L from BASIC.
   !   Several binary numbers are read and then converted to their
   !   integer equivalents.
   !-

   !+
   !   DECLARATIONS
   !-

   DECLARE STRING BIN_STR
   DECLARE LONG BIN_VAL, I, RET_STATUS
   DECLARE LONG CONSTANT FLAGS = 17        ! 2^0 + 2^4
   EXTERNAL LONG FUNCTION OTS$CVT_TB_L (STRING, LONG, &
       LONG BY VALUE, LONG BY VALUE)

   !+
   !   MAIN PROGRAM
   !-

   !+
   !   Read the data, convert it to binary, and print the result.
   !-

   FOR I = 1 TO 5
       READ BIN_STR
       RET_STATUS = OTS$CVT_TB_L( BIN_STR, BIN_VAL, '4'L, FLAGS)
       PRINT BIN_STR;" treated as a binary number equals";BIN_VAL
   NEXT I

   !+
   !   Done, end the program.
   !-

   GOTO 32767
```

```
999 Data    "1111", "1 111", "1011011", "11111111", "00000000"
32767 END
```

This BASIC example program demonstrates how to call OTS$CVT_TB_L to convert binary text to a longword integer.

The output generated by this BASIC program is as follows:

```
1111 treated as a binary number equals 15
1 111 treated as a binary number equals 15
1011011 treated as a binary number equals 91
11111111 treated as a binary number equals 255
00000000 treated as a binary number equals 0
```

OTS$CVT_TI_L—Convert Signed Integer Text to Integer

OTS$CVT_TI_L converts an ASCII text string representation of a decimal number to a signed byte, word, or longword integer value. The result is a longword by default, but the calling program can specify a byte or a word value instead.

FORMAT

OTS$CVT_TI_L *inp-str ,value [,value-size] [,flags]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

inp-str

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor, fixed-length or dynamic string**

Input ACSII text string that OTS$CVT_TI_L converts to a signed byte, word, or longword. The **inp-str** argument is the address of a descriptor pointing to the input string.

The syntax of a valid ASCII text input string is as follows:

```
[+ or -] <integer-digits>
```

OTS$CVT_TI_L always ignores leading blanks. A decimal point is assumed at the right of the input string.

value

VMS Usage: **varying_arg**
type: **unspecified**
access: **write only**
mechanism: **by reference**

Signed byte, word, or longword integer value (depending on **value-size**) that OTS$CVT_TI_L creates when it converts the ASCII text string. The **value** argument is the address of the integer value.

value-size

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by value**

Number of bytes occupied by the value that OTS$CVT_TI_L creates when it converts the ASCII text string to an integer value. Valid values for the **value-size** argument are 1, 2, and 4. The contents of **value-size** determine whether the integer value that OTS$CVT_TI_L creates is a byte, word, or longword. If an invalid value is given, OTS$CVT_TI_L returns an error. This is an

optional argument. If omitted, the default is 4 and OTS$CVT_TI_L returns a longword integer.

flags

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

User-supplied flags that OTS$CVT_TI_L uses to determine how blanks and tabs are interpreted. The **flags** argument is an unsigned longword containing the value of the flags.

Bit	Description
0	If set, OTS$CVT_TI_L ignores all blanks. If clear, OTS$CVT_TI_L ignores leading blanks but interprets blanks after the first legal character as zeros.
4	If set, OTS$CVT_TI_L ignores tabs. If clear, OTS$CVT_TI_L interprets tabs as invalid characters.

This is an optional argument. If omitted, the default is that all bits are cleared and OTS$CVT_TI_L ignores blanks and tabs.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
OTS$_INPCONERR	Input conversion error; an invalid character in the input string, or the value overflows byte, word, or longword, or **value-size** is invalid; **value** is set to zero.

OTS$CVT_TL_L—Convert Logical Text to Integer

OTS$CVT_TL_L converts an ASCII text string representation of a FORTRAN-77 L format to a byte, word, or longword integer value. The result is a longword by default, but the calling program can specify a byte or a word value instead.

FORMAT

OTS$CVT_TL_L *inp-str ,value [,value-size]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

inp-str

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor, fixed-length or dynamic string**

Input string containing an ASCII text representation of a FORTRAN-77 L format that OTS$CVT_TL_L converts to a byte, word or longword integer value. The **inp-str** argument is the address of a descriptor pointing to the input string.

The syntax of a valid ASCII text input string is as follows:

```
                <zero or more blanks>
                <     <end of string>
                              or
                      <       <"." or nothing>
Letter:                              <"T", "t", "F", or "f">
                              <zero or more of any character>
                              <end of string>>>
```

value

VMS Usage: **varying_arg**
type: **unspecified**
access: **write only**
mechanism: **by reference**

Integer value that OTS$CVT_TL_L creates when it converts the ACSII text input string. The **value** argument is the address of this integer value. OTS$CVT_TL_L returns a minus one (-1) as the contents of the **value** argument if the character denoted by "Letter:" is "T" or "t". Otherwise, OTS$CVT_TL_L sets **value** to zero.

value-size

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by value**

Number of bytes occupied by the integer value that OTS$CVT_TL_L creates when it converts the ASCII text input string. The **value-size** argument contains the number of bytes. Valid values are 1, 2, and 4. These values determine whether OTS$CVT_TL_L returns a byte, word or longword integer value. If an invalid value is given, OTS$CVT_TL_L returns an error. This is an optional argument. If omitted, the default is 4 and OTS$CVT_TL_L returns a longword integer value.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
OTS$_INPCONERR	Invalid character in the input string or invalid **value-size**; **value** is set to zero.

OTS$CVT_TO_L—Convert Octal Text to Signed Integer

OTS$CVT_TO_L converts an ASCII text string representation of an unsigned octal value to an unsigned integer of an arbitrary length. The result is a longword by default, but the calling program can specify any number of bytes.

FORMAT

OTS$CVT_TO_L *inp-str ,value [,value-size] [,flags]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

inp-str

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor, fixed-length or dynamic string**

Input string containing an ASCII text string representation of an unsigned octal value that OTS$CVT_TO_L converts to an unsigned integer. The **inp-str** argument is the address of a descriptor pointing to the input string. The valid input characters are blanks and the digits 0 through 7. No sign is permitted.

value

VMS Usage: **varying_arg**
type: **unspecified**
access: **write only**
mechanism: **by reference**

Integer value that OTS$CVT_TO_L creates when it converts the input string. The **value** argument is the address of the unsigned integer value.

value-size

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by value**

Number of bytes occupied by the unsigned integer value. The **value-size** argument contains the number of bytes. If the content of the **value-size** argument is zero or a negative number, OTS$CVT_TO_L returns an error. This is an optional argument. If omitted, the default is 4 and OTS$CVT_TO_L returns a longword integer.

flags

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

User-supplied flags that OTS$CVT_TO_L uses to determine how blanks within the input string are interpreted. The **flags** argument contains the user-supplied flags.

Bit 0 — If set, OTS$CVT_TO_L ignores all blanks. If clear, OTS$CVT_TO_L interprets blanks as zeros.

This is an optional argument. If omitted, the default is that all bits are clear.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
OTS$_INPCONERR	Input conversion error. An invalid character, overflow, or invalid **value-size** occurred.

EXAMPLE

```
OCTAL_CONV: PROCEDURE OPTIONS (MAIN) RETURNS (FIXED BINARY (31));
%INCLUDE $STSDEF;       /* Include definition of return status values      */
DECLARE OTS$CVT_TO_L ENTRY
        (CHARACTER (*),         /* Input string passed by descriptor        */
        FIXED BINARY (31),      /* Returned value passed by reference       */
        FIXED BINARY VALUE,     /* Size for returned value passed by value  */
        FIXED BINARY VALUE)     /* Flags passed by value                    */
        RETURNS (FIXED BINARY (31))     /* Return status                    */
        OPTIONS (VARIABLE);     /* Arguments may be omitted                 */
DECLARE INPUT CHARACTER (10);
DECLARE VALUE FIXED BINARY (31);
DECLARE SIZE  FIXED BINARY(31) INITIAL(4) READONLY STATIC; /* Longword      */
DECLARE FLAGS FIXED BINARY(31) INITIAL(1) READONLY STATIC; /* Ignore blanks */
ON ENDFILE (SYSIN) STOP;
DO WHILE ('1'B);                /* Loop continuously, until end of file    */
        PUT SKIP (2);
        GET LIST (INPUT) OPTIONS (PROMPT ('Octal value: '));
        STS$VALUE = OTS$CVT_TO_L (INPUT, VALUE, SIZE, FLAGS);
        IF ^STS$SUCCESS THEN RETURN (STS$VALUE);
        PUT SKIP EDIT (INPUT, 'Octal equals', VALUE, 'Decimal')
                    (A,X,A,X,F(10),X,A);
        END;
END OCTAL_CONV;
```

This PL/I program translates an octal value in ASCII into a fixed binary value. The program is run interactively; simply type CTRL/Z to quit.

```
$ RUN OCTOL
Octal value: 1
1  Octal equals 1 Decimal
Octal value: 11
11   Octal equals   9 Decimal
Octal value: 1017346
1017346 Octal equals 274150 Decimal
Octal value: CTRL/Z
```

OTS$CVT_TU_L—Convert Unsigned Decimal Text to Integer

OTS$CVT_TU_L converts an ASCII text string representation of an unsigned decimal value to an unsigned byte, word, or longword value. By default, the result is a longword. Valid input characters are the space and the digits 0 through 9. No sign is permitted.

FORMAT

OTS$CVT_TU_L *inp-str ,value [,value-size] [,flags]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

inp-str

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor, fixed-length**

Input string (fixed-length) containing an ASCII text string representation of an unsigned decimal value that OTS$CVT_TU_L converts to a byte, word, or longword value. The **inp-str** argument is the address of a descriptor pointing to the input string.

value

VMS Usage: **varying_arg**
type: **unspecified**
access: **write only**
mechanism: **by reference**

Byte, word, or longword (depending on **value-size**) into which OTS$CVT_TU_L writes the converted value. The **value** argument is the address of the byte, word, or longword.

value-size

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by value**

Number of bytes occupied by the value created when OTS$CVT_TU_L converts the input string. The **value-size** argument contains the number of bytes. OTS$CVT_TU_L allows value sizes of 1, 2 and 4. If any other value is specified, or if **value-size** is omitted, OTS$CVT_TU_L uses the default, 4.

flags

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

User-supplied flags which OTS$CVT_TU_L uses to determine how blanks and tabs are interpreted. The **flags** argument contains the user-supplied flags.

Bit	Description
0	If set, OTS$CVT_TU_L ignores blanks. If clear, OTS$CVT_TU_L interprets blanks as zeros.
4	If set, OTS$CVT_TU_L ignores tabs. If clear, OTS$CVT_TU_L interprets tabs as invalid characters.

Flags is an optional argument. If omitted, the default is that all bits are clear.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
OTS$_INPCONERR	Input conversion error. An invalid character, overflow or invalid **value-size** occurred.

OTS$CVT_T_z—Convert Numeric Text to Floating

The OTS$CVT_T_z routines convert an ASCII text string representation of a numeric value to a D_floating, F_floating, G_floating, or H_floating value.

FORMAT

OTS$CVT_T_D *inp-str ,value [,digits-in-fract] [,scale-factor] [,flags] [,ext-bits]*
OTS$CVT_T_F *inp-str ,value [,digits-in-fract] [,scale-factor] [,flags] [,ext-bits]*
OTS$CVT_T_G *inp-str ,value [,digits-in-fract] [,scale-factor] [,flags] [,ext-bits]*
OTS$CVT_T_H *inp-str ,value [,digits-in-fract] [,scale-factor] [,flags] [,ext-bits]*

Each of the above four formats corresponds to one of the four floating-point types.

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

inp-str

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor, fixed-length or dynamic string**

Input string containing an ASCII text string representation of a numeric value that OTS$CVT_T_Z converts to a D_floating, F_floating, G-floating, or H-floating value. The **inp-str** argument is the address of a descriptor pointing to the input string.

The syntax of a valid input string is as follows:

```
<zero or more blanks>
<"+", "-", or nothing>
<zero or more decimal digits>
<"." or nothing>
<zero or more decimal digits>
<exponent or nothing, where exponent is:
        <    <<"E", "e", "D", "d", "Q", or "q">
                <zero or more blanks>
                <"+", "-", or nothing>>
                or
                <"+", or "-">>
        <zero or more decimal digits>>
<end of string>
```

There is no difference in semantics among any of the six valid exponent letters (E, e, D, d, Q, q).

value

VMS Usage: **floating_point**
type: **D_floating, F_floating, G_floating, H_floating**
access: **write only**
mechanism: **by reference**

Floating-point value that OTS$CVT_T_z creates when it converts the input string. The **value** argument is the address of the floating-point value. For OTS$CVT_T_D, **value** is a D_floating number. For OTS$CVT_T_F, **value** is an F_floating number. For OTS$CVT_T_G, **value** is a G_floating number. For OTS$CVT_T_H, **value** is an H_floating number.

digits-in-fract

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Number of digits in the fraction if no decimal point is included in the input string. The **digits-in-fract** argument contains the number of digits. This is an optional argument. If omitted, the default is zero.

scale-factor

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by value**

Scale factor. The **scale-factor** argument contains the value of the scale factor. If bit 6 of the **flags** argument is clear, the resultant value is divided by 10^{factor} unless the exponent is present. If bit 6 of **flags** is set, the scale factor is always applied. This is an optional argument. If omitted, the default is zero.

flags

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

User-supplied flags. The **flags** argument contains the user-supplied flags.

Bit 0 — If set, OTS$CVT_T_z ignores blanks. If clear, OTS$CVT_T_z interprets blanks as zeros.

Bit 1 — If set, OTS$CVT_T_z allows only E or e exponents. If clear, OTS$CVT_T_z allows E, e, D, d, Q and q exponents. (Bit 1 is clear for BASIC and set for FORTRAN.)

Bit 2 — If set, OTS$CVT_T_z interprets an underflow as an error. If clear, OTS$CVT_T_z does not interpret an underflow as an error.

Bit 3 — If set, OTS$CVT_T_z truncates the value. If clear, OTS$CVT_T-z rounds the value.

Bit 4 — If set, OTS$CVT_T_z ignores tabs. If clear, OTS$CVT_T_z interprets tabs as invalid characters.

Bit 5	If set, an exponent must begin with a valid exponent letter. If clear, the exponent letter may be omitted.
Bit 6	If set, OTS$CVT_T_z always applies the scale factor. If clear, OTS$CVT_T_z applies the scale factor only if there is no exponent present in the string.

If **flags** is omitted, all bits are clear.

ext-bits

VMS Usage: **word_signed**
type: **word integer (signed)**
access: **write only**
mechanism: **by reference**

Extra precision bits. The **ext-bits** argument is the address of a signed word integer containing the extra precision bits. If present, **value** is not rounded, and the first n bits after truncation are returned in this argument. For D_floating and F_floating, n equals 8 and the bits are returned as a byte. For G_floating and H_floating, n equals 11 and 15, respectively, and the bits are returned as a word, left-justified.

These values are suitable for use as the extension operand in an EMOD instruction.

The extra precision bits returned for H_floating may not be precise because calculations are only carried to 128 bits. However, the error should be small.

DESCRIPTION

These routines support FORTRAN D, E, F, and G input type conversion as well as similar types for other languages.

OTSCVT_T_D, OTSCVT_T_F, OTS$CVT_T_G, and OTS$CVT_T_H provide run-time support for BASIC and FORTRAN input statements.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
OTS$_INPCONERR	Input conversion error; an invalid character in the input string, or the value is outside the range that can be represented. **Value** is set to +0.0 (not reserved operand -0.0).

EXAMPLE

```
C+
C This is a FORTRAN program demonstrating the use of
C OTS$CVT_T_F.
C-
        REAL*4 A
        CHARACTER*10 T(5)
        DATA T/'1234567+23','8.786534+3','-983476E-3','-23.734532','45'/
        DO 2 I = 1, 5
        TYPE 1,I,T(I)
 1      FORMAT(' Input string ',I1,' is ',A10)
C+
C B is the return status.
C T(I) is the string to be converted to an
C F_floating point value.  A is the F_floating
C point conversion of T(I).  %VAL(5) means 5 digits
C are in the fraction if no decimal point is in
```

```
C the input string T(I).
C-
        B = OTS$CVT_T_F(T(I),A,%VAL(5),,)
        TYPE *,' Output of OTSCVT_T_F is     ',A
        TYPE *,'  '
 2      CONTINUE
        END
```

This FORTRAN example demonstrates the use of OTS$CVT_T_F. The output generated by this program is as follows:

```
Input string 1 is 1234567+23
 Output of OTSCVT_T_F is       1.2345669E+24
Input string 2 is 8.786534+3
 Output of OTSCVT_T_F is        8786.534
Input string 3 is -983476E-3
 Output of OTSCVT_T_F is      -9.8347599E-03
Input string 4 is -23.734532
 Output of OTSCVT_T_F is       -23.73453
Input string 5 is 45
 Output of OTSCVT_T_F is        45000.00
```

OTS$CVT_TZ_L—Convert Hexadecimal Text to Unsigned Integer

OTS$CVT_TZ_L converts an ASCII text string representation of an unsigned hexadecimal value to an unsigned integer of an arbitrary length. The result is a longword by default, but the calling program can specify either 1, 2 or 4 bytes to receive either a byte, word, or longword value.

FORMAT

OTS$CVT_TZ_L *inp-str ,value [,value-size] [,flags]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

inp-str

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor, fixed-length or dynamic string**

Input string containing an ASCII text string representation of an unsigned hexadecimal value that OTS$CVT_TZ_L converts to an unsigned integer. The **inp-str** argument is the address of a descriptor pointing to the input string. Valid input characters are the space, the digits 0 through 9, and the letters A through F. No sign is permitted. Lowercase letters a through f are acceptable.

value

VMS Usage: **varying_arg**
type: **unspecified**
access: **write only**
mechanism: **by reference**

Integer value created when OTS$CVT_TZ_L converts the input string. The **value** argument is the address of the integer value.

value-size

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by value**

Number of bytes occupied by the integer value. The **value-size** argument contains the number of bytes. If the value size is zero or a negative number, OTS$CVT_TZ_L returns an input conversion error. This is an optional argument. If omitted, the default is 4.

flags

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

User-supplied flags that OTS$CVT_TZ_L uses to determine how blanks are interpreted. The **flags** argument is an unsigned longword containing these user-supplied flags.

Bit 0	If set, OTS$CVT_TZ_L ignores blanks. If set, OTS$CVT_TZ_L interprets blanks as zeros.

This is an optional argument. If omitted, the default is that all bits are clear.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
OTS$_INPCONERR	Input conversion error. An invalid character, overflow, or invalid **value-size** occurred.

EXAMPLES

1

```
10      !+
        ! This BASIC program converts a character string representing
        ! a hexadecimal value to a longword.
        !-
100     !+
        ! Illustrate (and test) OTS convert hex-string to longword
        !-
        EXTERNAL LONG FUNCTION OTS$CVT_TZ_L
        EXTERNAL LONG CONSTANT OTS$_INPCONERR
        INPUT "Enter hex numeric";HEXVAL$
        RET_STAT% = OTS$CVT_TZ_L(HEXVAL$, HEX% )
        PRINT "Conversion error " IF RET_STAT% = OTS$_INPCONERR
        PRINT "Decimal value of ";HEXVAL$;" is";HEX%             &
            IF RET_STAT% <> OTS$_INPCONERR
```

This BASIC example accepts a hexadecimal numeric string, converts it to a decimal integer, and prints the result. One sample of the output generated by this program is as follows:

```
$ RUN HEX
Enter hex numeric? A
Decimal value of A is 10
```

2

```
HEX_CONV: PROCEDURE OPTIONS (MAIN) RETURNS (FIXED BINARY (31));
%INCLUDE $STSDEF;       /* Include definition of return status values      */
DECLARE OTS$CVT_TZ_L ENTRY
        (CHARACTER (*),         /* Input string passed by descriptor       */
        FIXED BINARY (31),      /* Returned value passed by reference      */
        FIXED BINARY VALUE,     /* Size for returned value passed by value */
        FIXED BINARY VALUE)     /* Flags passed by value                   */
        RETURNS (FIXED BINARY (31))     /* Return status                   */
        OPTIONS (VARIABLE);     /* Arguments may be omitted                */
DECLARE INPUT CHARACTER (10);
DECLARE VALUE FIXED BINARY (31);
DECLARE FLAGS FIXED BINARY(31) INITIAL(1) READONLY STATIC; /* Ignore blanks */
ON ENDFILE (SYSIN) STOP;
DO WHILE ('1'B);                /* Loop continuously, until end of file    */
        PUT SKIP (2);
        GET LIST (INPUT) OPTIONS (PROMPT ('Hex value: '));
```

```
        STS$VALUE = OTS$CVT_TZ_L (INPUT, VALUE, , FLAGS);
        IF ^STS$SUCCESS THEN RETURN (STS$VALUE);
        PUT SKIP EDIT (INPUT, 'Hex equals', VALUE, 'Decimal')
                    (A,X,A,X,F(10),X,A);
        END;
END HEX_CONV;
```

This PL/I example translates a hexadecimal value in ASCII into a fixed binary value. This program continues to prompt for input values until the user types CTRL/Z.

One sample of the output generated by this program is as follows:

```
$ RUN HEX
Hex value: 1A
1A      Hex equals       26 Decimal
Hex value: C
C       Hex equals       12 Decimal
Hex value: CTRL/Z
```

OTS$DIVCx—Complex Division

OTS$DIVC, OTS$DIVCD_R3 and OTS$DIVCG_R3 return a complex result of a complex division on complex numbers.

FORMAT

OTS$DIVC *dividend ,divisor*
OTS$DIVCD_R3 *dividend ,divisor*
OTS$DIVCG_R3 *dividend ,divisor*

Each of the above three formats corresponds to one of the three floating-point complex types.

RETURNS

VMS Usage:	**complex_number**
type:	**F_floating complex**
access:	**write only**
mechanism:	**by value**

Complex result of complex division. OTS$DIVC returns an F-floating number. OTS$DIVCD_R3 returns a D_floating number. OTS$DIVCG_R3 returns a G_floating number.

ARGUMENTS

dividend

VMS Usage:	**complex_number**
type:	**F_floating complex, D_floating complex, G_floating complex**
access:	**read only**
mechanism:	**by value**

Complex dividend. The **dividend** argument contains a floating-point complex value. For OTS$DIVC, **dividend** is an F_floating complex number. For OTS$DIVCD_R3, **dividend** is a D_floating complex number. For OTS$DIVCG_R3, **dividend** is a G_floating complex number.

divisor

VMS Usage:	**complex_number**
type:	**F_floating complex, D_floating complex, G_floating complex**
access:	**read only**
mechanism:	**by value**

Complex divisor. The **divisor** argument contains the value of the divisor. For OTS$DIVC, **divisor** is an F_floating complex number. For OTS$DIVCD_R3, **divisor** is a D_floating complex number. For OTS$DIVCG_R3, **divisor** is a G_floating complex number.

DESCRIPTION

These procedures return a complex result of a complex division on complex numbers.

The complex result is computed as follows:

1 Let (a,b) represent the complex dividend.
2 Let (c,d) represent the complex divisor.
3 Let (r,i) represent the complex quotient.

The results of this computation are as follows:

```
r = (ac + bd)/(cc + dd)
i = (bc - ad)/(cc + dd)
```

CONDITION VALUES SIGNALED

SS$_FLTDIV_F	Arithmetic fault. Floating-point division by zero.
SS$_FLTOV_F	Arithmetic fault. Floating-point overflow.

EXAMPLES

1

```
C+
C       This FORTRAN example forms the complex
C       quotient of two complex numbers using
C       OTS$DIVC and the FORTRAN random number
C       generator RAN.
C
C       Declare Z1, Z2, Z_Q, and OTS$DIVC as complex values.
C       OTS$DIVC will return the complex quotient of Z1 divided
C       by Z2:  Z_Q = OTS$DIVC( %VAL(REAL(Z1)), %VAL(AIMAG(Z1),
C       %VAL(REAL(Z2)), %VAL(AIMAG(Z2))
C-
          COMPLEX Z1,Z2,Z_Q,OTS$DIVC
C+
C       Generate a complex number.
C-
          Z1 = (8.0,4.0)
C+
C       Generate another complex number.
C-
          Z2 = (1.0,1.0)
C+
C       Compute the complex quotient of Z1/Z2.
C-
          Z_Q = OTS$DIVC( %VAL(REAL(Z1)), %VAL(AIMAG(Z1)), %VAL(REAL(Z2)),
     +                    %VAL(AIMAG(Z2)))
          TYPE *, ' The complex quotient of',Z1,' divided by ',Z2,' is:'
          TYPE *, '      ',Z_Q
          END
```

This FORTRAN program demonstrates how to call OTS$DIVC. The output generated by this program is as follows:

```
The complex quotient of (8.000000,4.000000) divided by  (1.000000,1.000000) is:
        (6.000000,-2.000000)
```

2

```
C+
C    This FORTRAN example forms the complex
C    quotient of two complex numbers by using
C    OTS$DIVCG_R3 and the FORTRAN random number
C    generator RAN.
C
C     Declare Z1, Z2, and Z_Q as complex values. OTS$DIVCG_R3
C     will return the complex quotient of Z1 divided by Z2:
C     Z_Q = Z1/Z2
C-
        COMPLEX*16 Z1,Z2,Z_Q
C+
C    Generate a complex number.
C-
        Z1 = (8.0,4.0)
C+
C    Generate another complex number.
C-
        Z2 = (1.0,1.0)
C+
C    Compute the complex quotient of Z1/Z2.
C-
        Z_Q = Z1/Z2
        TYPE *, ' The complex quotient of',Z1,' divided by ',Z2,' is:'
        TYPE *, '     ',Z_Q
        END
```

This FORTRAN example uses the OTS$DIVCG_R3 entry point instead. Notice the difference in the precision of the output generated:

```
 The complex quotient of (8.000000000000000,4.000000000000000) divided by
(1.000000000000000,1.000000000000000) is:
      (6.000000000000000,-2.000000000000000)
```

OTS$DIV_PK_LONG—Packed Decimal Division with Long Divisor

OTS$DIV_PK_LONG divides fixed-point decimal data, which is stored in packed decimal form, when precision and scale requirements for the quotient call for multiple precision division. The divisor must have a precision of thirty or thirty-one digits.

FORMAT

OTS$DIV_PK_LONG *divd ,divr ,divr-prec ,quot ,quot-prec ,prec-data ,scale-data*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

divd

VMS Usage: **address**
type: **packed decimal string**
access: **read only**
mechanism: **by reference**

Dividend. The **divd** argument is the address of a packed decimal string which contains the shifted dividend.

The **divd** argument is always multiplied by (10**c) prior to passing it as input, where c is defined as follows:

```
c = 31 - prec(divd)
```

Mutiplying **divd** by (10**c) makes **divd** a 31 digit number.

divr

VMS Usage: **address**
type: **packed decimal string**
access: **read only**
mechanism: **by reference**

Divisor. The **divr** argument is the address of a packed decimal string which contains the divisor.

divr-prec

VMS Usage: **word_signed**
type: **word integer (signed)**
access: **read only**
mechanism: **by value**

Precision of the divisor. The **divr-prec** argument is a signed word integer which contains the precision of the divisor. The high order bits are filled with zeros.

quot

VMS Usage: **address**
type: **packed decimal string**
access: **write only**
mechanism: **by reference**

Quotient. The **quot** argument is the address of the packed decimal string into which OTS$DIV_PK_LONG writes the quotient.

quot-prec

VMS Usage: **word_signed**
type: **word integer (signed)**
access: **read only**
mechanism: **by value**

Precision of the quotient. The **quot-prec** argument is a signed word integer that contains the precision of the quotient. The high order bits are filled with zeros.

prec-data

VMS Usage: **word_signed**
type: **word integer (signed)**
access: **read only**
mechanism: **by value**

Additional digits of precision required. The **prec-data** argument is a signed word integer that contains the value of the additional digits of precision required.

OTS$DIV_PK_LONG computes the **prec-data** argument as follows:

```
prec-data = scale(quot) + scale(divr) - scale(divd) - 31 +
            prec(divd)
```

scale-data

VMS Usage: **word_signed**
type: **word integer (signed)**
access: **read only**
mechanism: **by value**

The **scale-data** argument is a signed word integer that contains the scale data.

OTS$DIV_PK_LONG defines the **scale-data** argument as follows:

```
scale-data = 31 - prec(divr)
```

DESCRIPTION

Before using this procedure, you should determine whether it would be best to use OTSDIV_PK_LONG, OTSDIV_PK_SHORT, or the VAX instruction DIVP. To determine this, you must first calculate *b*, where *b* is defined as follows:

```
b = scale(quot) + scale(divr) - scale(divd) +
    prec(divd)
```

If *b* is greater than 31, then OTS$DIV_PK_LONG may be used to perform the division. If *b* is less that 31, you could use the instruction DIVP instead.

Once you have determined that you cannot use DIVP, you need to determine whether you shuld use OTS$DIV_PK_LONG or OTS$DIV_PK_SHORT. To determine this, you must examine the value of **scale-data**. If **scale-data** is less than or equal to 1, then you should use OTS$DIV_PK_LONG. If **scale-data** is greater than 1, you should use OTS$DIV_PK_SHORT instead.

CONDITION VALUE SIGNALED

SS$_FLTDIV   Fatal error. Division by zero.

EXAMPLE

1

```
OPTION                                  &
    TYPE = EXPLICIT
!+
!   This program uses OTS$DIV_PK_LONG to perform packed decimal
!   arithmetic.
!-

!+
!   DECLARATIONS
!-
DECLARE DECIMAL (31, 2)     NATIONAL_DEBT
DECLARE DECIMAL (30, 3)     POPULATION
DECLARE DECIMAL (10, 5)     PER_CAPITA_DEBT
EXTERNAL SUB OTS$DIV_PK_LONG (DECIMAL(31,2), DECIMAL (30, 3), &
    WORD BY VALUE, DECIMAL(10, 5), WORD BY VALUE, WORD BY VALUE, &
    WORD BY VALUE)
!+
!   Prompt the user for the required input.
!-
INPUT   "Enter national debt: ";NATIONAL_DEBT
INPUT   "Enter current population: ";POPULATION

!+
!   Perform the division and print the result.
!
!   scale(divd) = 2
!   scale(divr) = 3
!   scale(quot) = 5
!
!   prec(divd) = 31
!   prec(divr) = 30
!   prec(quot) = 10
!
!   prec-data  = scale(quot) + scale(divr) - scale(divd) - 31 +
!                prec(divd)
!   prec-data  =   5       +   3       -     2    - 31 +   31
!   prec-data  = 6
!
!   b = scale(quot) + scale(divr) - scale(divd) + prec(divd)
!   b =    5       +   3       -      2     +     31
!   b = 37
!
!   c = 31 - prec(divd)
!   c = 31 -    31
!   c = 0
!
!   scale-data = 31 - prec(divr)
!   scale-data = 31 -   30
!   scale-data = 1
!
!   b is greater than 31, so either OTS$DIV_PK_LONG or
!      OTS$DIV_PK_SHORT may be used to perform the division.
!      If b is less than or equal to 31, then the DIVP
!      instruction may be used.
!
!   scale-data is less than or equal to 1, so OTS$DIV_PK_LONG
!      should be used instead of OTS$DIV_PK_SHORT.
```

```
!
!-
CALL OTS$DIV_PK_LONG( NATIONAL_DEBT, POPULATION, '30'W, PER_CAPITA_DEBT, &
        '10'W, '6'W, '1'W)
PRINT   "The per capita debt is: ";PER_CAPITA_DEBT
END
```

This BASIC example program uses OTS$DIV_PK_LONG to perform packed decimal division. One example of the output generated by this program is as follows:

```
$ RUN DEBT
Enter national debt: ?  12345678
Enter current population: ?  1212
The per capita debt is: 10186.20297
```

OTS$DIV_PK_SHORT—Packed Decimal Division with Short Divisor

OTS$DIV_PK_SHORT divides fixed-point decimal data, which is stored in packed decimal form, when precision and scale requirements for the quotient call for multiple-precision division. The divisor can have a maximum precision of twenty-nine digits.

FORMAT

OTS$DIV_PK_SHORT *divd ,divr ,divr-prec ,quot ,quot-prec ,prec-data*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

divd

VMS Usage: **address**
type: **packed decimal string**
access: **read only**
mechanism: **by reference**

Dividend.The **divd** argument is the address of a packed decimal string which contains the shifted dividend.

The **divd** argument is always multiplied by (10**c) prior to passing it as input, where c is defined as:

```
c = 31 - prec(divd)
```

Multiplying **divd** by (10**c) makes **divd** a 31 digit number.

divr

VMS Usage: **address**
type: **packed decimal string**
access: **read only**
mechanism: **by reference**

Divisor. The **divr** argument is the address of a packed decimal string which contains the divisor.

divr-prec

VMS Usage: **word_signed**
type: **word integer (signed)**
access: **read only**
mechanism: **by value**

Precision of the divisor. The **divr-prec** argument is a signed word integer which contains the precision of the divisor. The high-order bits are filled with zeros.

quot

VMS Usage: **address**
type: **packed decimal string**
access: **write only**
mechanism: **by reference**

Quotient. The **quot** argument is the address of a packed decimal string into which OTS$DIV_PK_SHORT writes the quotient.

quot-prec

VMS Usage: **word_signed**
type: **word integer (signed)**
access: **read only**
mechanism: **by value**

Precision of the quotient. The **quot-prec** argument is a signed word integer which contains the precision of the quotient. The high-order bits are filled with zeros.

prec-data

VMS Usage: **word_signed**
type: **word integer (signed)**
access: **read only**
mechanism: **by value**

Additional digits of precision required. The **prec-data** argument is a signed word integer which contains the value of the additional digits of precision required.

OTS$DIV_PK_SHORT computes the **prec-data** argument as follows:

```
prec-data = scale(quot) + scale(divr) - scale(divd) - 31 +
            prec(divd)
```

DESCRIPTION

Before using this procedure, you should determine whether it would be best to use OTSDIV_PK_LONG, OTSDIV_PK_SHORT, or the VAX instruction DIVP. To determine this, you must first calculate *b*, where *b* is defined as follows:

```
b = scale(quot) + scale(divr) - scale(divd) + prec(divd)
```

If *b* is greater than 31, then OTS$DIV_PK_SHORT may be used to perform the division. If *b* is less that 31, you could use the VAX instruction DIVP instead.

Once you have determined that you cannot use DIVP, you need to determine whether you should use OTS$DIV_PK_LONG or OTS$DIV_PK_SHORT. To determine this, you must examine the value of **scale-data**. If **scale-data** is less than or equal to 1, then you should use OTS$DIV_PK_LONG. If **scale-data** is greater than 1, you should use OTS$DIV_PK_SHORT instead.

CONDITION VALUE SIGNALED

SS$_FLTDIV	Fatal error. Division by zero.

OTS$MOVE3 - Move Data Without Fill

OTS$MOVE3 moves up to 2**31-1 bytes, (2,147,483,647 bytes) from a specified source address to a specified destination address.

FORMAT

OTS$MOVE3 *length ,source ,dest*

corresponding jsb entry point

OTS$MOVE3_R5

RETURNS

None.

ARGUMENTS

length

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by value**

Number of bytes of data to move. The **length** argument contains the number of bytes to move. The value of **length** may range from 0 to 2,147,483,647 bytes.

source

VMS Usage: **byte_unsigned**
type: **byte (unsigned)**
access: **read only**
mechanism: **by reference, array reference**

Data to be moved by OTS$MOVE3. The **source** argument contains the address of an unsigned byte array that contains this data.

dest

VMS Usage: **byte_unsigned**
type: **byte (unsigned)**
access: **write only**
mechanism: **by reference, array reference**

Item into which **source** will be moved. The **dest** argument is the address of an unsigned byte array into which OTS$MOVE3 writes the source data.

DESCRIPTION

OTS$MOVE3 performs the same function as the VAX MOVC3 instruction except that the **length** is a longword integer rather than a word integer. When called from the JSB entry point, the register outputs of OTS$MOVE3_R5 follow the same pattern as those of the MOVC3 instruction:

R0	0
R1	Address of one byte beyond the source string
R2	0
R3	Address of one byte beyond the destination string
R4	0
R5	0

For more information, see the description of the MOVC3 instruction in the *VAX-11 Architecture Reference Manual.* See also the routine LIB$MOVC3, which is a callable version of the MOVC3 instruction.

CONDITION VALUES RETURNED

None.

OTS$MOVE5—Move Data with Fill

OTS$MOVE5 moves up to 2**31-1 bytes, (2,147,483,647 bytes) from a specified source address to a specified destination address, with separate source and destination lengths, and with fill. Overlap of the source and destination arrays does not affect the result.

FORMAT

OTS$MOVE5 *srclen ,source ,fill ,dstlen ,dest*

corresponding jsb entry point

OTS$MOVE5_R5

RETURNS

None.

ARGUMENTS

srclen

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by value**

Number of bytes of data to move. The **srclen** argument contains a signed longword integer that is this number. The value of **srclen** may range from 0 to 2,147,483,647.

source

VMS Usage: **byte_unsigned**
type: **byte (unsigned)**
access: **read only**
mechanism: **by reference, array reference**

Data to be moved by OTS$MOVE5. The **source** argument contains the address of an unsigned byte array that contains this data.

fill

VMS Usage: **byte_unsigned**
type: **byte (unsigned)**
access: **read only**
mechanism: **by value**

Character used to pad the source data if **scrclen** is less than **dstlen**. The **fill** argument contains the address of an unsigned byte that is this character.

dstlen

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by value**

Size of the destination area in bytes. The **dstlen** argument is a signed longword integer containing this size. The value of **dstlen** may range from 0 through 2,147,483,647.

dest

VMS Usage: **byte_unsigned**
type: **byte (unsigned)**
access: **write only**
mechanism: **by reference, array reference**

Item into which **source** will be moved. The **dest** argument is the address of an unsigned byte array into which OTS$MOVE5 will write the source data.

DESCRIPTION

OTS$MOVE5 performs the same function as the VAX MOVC5 instruction except that the **srclen** and **dstlen** arguments are longword integers rather than word integers. When called from the JSB entry point, the register outputs of OTS$MOVE5_R5 follow the same pattern as those of the MOVC5 instruction:

R0	Number of unmoved bytes remaining in source string
R1	Address of one byte beyond the source string
R2	0
R3	Address of one byte beyond the destination string
R4	0
R5	0

For more information, see the description of the MOVC5 instruction in the *VAX-11 Architecture Reference Manual*. See also the routine LIB$MOVC5, which is a callable version of the MOVC5 instruction.

CONDITION VALUES RETURNED

None.

OTS$MULCx—Complex Multiplication

OTS$MULCD_R3 and OTS$MULCG_R3 calculate the complex product of two complex values.

FORMAT

OTS$MULCD_R3 *multiplier ,multiplicand*
OTS$MULCG_R3 *multiplier ,multiplicand*

The above formats correspond to the D_floating and G_floating complex types.

RETURNS

VMS Usage: **complex_number**
type: **D_floating complex, G_floating complex**
access: **write only**
mechanism: **by value**

Complex result of multiplying two complex numbers. OTS$MULCD_R3 returns a D_floating complex number. OTS$MULCG_R3 returns a G_floating complex number.

ARGUMENTS

multiplier

VMS Usage: **complex_number**
type: **D_floating complex, G_floating complex**
access: **read only**
mechanism: **by value**
Complex multiplier. The **multiplier** argument contains the multiplier. For OTS$MULCD_R3, **multiplier** is a D_floating complex number. For OTS$MULCG_R3, **multiplier** is a G_floating complex number.

multiplicand

VMS Usage: **complex_number**
type: **D_floating complex, G_floating complex**
access: **read only**
mechanism: **by value**
Complex multiplicand. The **multiplicand** argument contains the multiplicand. For OTS$MULCD_R3, **multiplicand** is a D_floating complex number. For OTS$MULCG_R3, **multiplicand** is an F_floating complex number.

DESCRIPTION

OTS$MULCD_R3 and OTS$MULCG_R3 calculate the complex product of two complex values.

The complex product is computed as follows:

1 Let (a,b) represent the complex multiplier.

2 Let (c,d) represent the complex multiplicand.

3 Let (r,i) represent the complex product.

The results of this computation are as follows:

```
r = ac - bd
i = ad + bc
```

CONDITION VALUES SIGNALED

MTH$_FLOOVEMAT	Floating-point overflow in Math Library.
SS$_ROPRAND	Reserved operand. OTS$MULCx encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.

EXAMPLE

```
C+
C    This FORTRAN example forms the product of
C    two complex numbers using OTS$MULCD_R3
C    and the FORTRAN random number generator RAN.
C
C    Declare Z1, Z2, and Z_Q as complex values. OTS$MULCD_R3
C    will return the complex product of Z1 times by Z2:
C    Z_Q = Z1 * Z2
C-
        COMPLEX*16 Z1,Z2,Z_Q
C+
C    Generate a complex number.
C-
        Z1 = (8.0,4.0)
C+
C    Generate another complex number.
C-
        Z2 = (2.0,3.0)
C+
C    Compute the complex product of Z1*Z2.
C-
        Z_Q = Z1 * Z2
        TYPE *, ' The complex product of',Z1,' times ',Z2,' is:'
        TYPE *, '     ',Z_Q
        END
```

This FORTRAN example uses OTS$MULCD_R3 to multiply two complex numbers. The output generated by this program is as follows:

```
 The complex product of (8.000000000000000,4.000000000000000) times
(2.000000000000000,3.000000000000000) is:
     (4.000000000000000,32.00000000000000)
```

OTS$POWCxCx—Raise a Complex Base to a Complex Floating-Point Exponent

OTS$POWCC, OTS$POWCDCD_R3 and OTS$POWCGCG_R3 return the result of raising a complex base to a complex exponent.

FORMAT

OTS$POWCC *base ,exponent*
OTS$POWCDCD_R3 *base ,exponent*
OTS$POWCGCG_R3 *base ,exponent*

Each of the above three formats corresponds to one of the three floating-point complex types.

RETURNS

VMS Usage: **complex_number**
type: **F_floating complex, D_floating complex, G_floating complex**
access: **write only**
mechanism: **by value**

Result of raising a complex base to a complex exponent. OTS$POWCC returns an F_floating complex number. OTS$POWCDCD_R3 returns a D_floating complex number. OTS$POWCGCG_R3 returns a G_floating complex number.

ARGUMENTS

base

VMS Usage: **complex_number**
type: **F_floating complex, D_floating complex, G_floating complex**
access: **read only**
mechanism: **by value**

Complex base. The **base** argument contains the value of the base. For OTS$POWCC, **base** is an F_floating complex number. For OTS$POWCDCD_R3, **base** is a D_floating complex number. For OTS$POWCGCG_R3, **base** is a G_floating complex number.

exponent

VMS Usage: **complex_number**
type: **F_floating complex, D_floating complex, G_floating complex**
access: **read only**
mechanism: **by value**

Complex exponent. The **exponent** argument contains the value of the exponent. For OTS$POWCC, **exponent** is an F_floating complex number. For OTS$POWCDCD_R3, **exponent** is a D_floating complex number. For OTS$POWCGCG_R3, **exponent** is a G_floating complex number.

DESCRIPTION

OTS$POWCC, OTS$POWCDCD_R3 and OTS$POWCGCG_R3 return the result of raising a complex base to a complex exponent. The American National Standard FORTRAN-77 (ANSI X3.9-1978) Standard defines complex exponentiation as:

```
X^Y = CEXP(Y * CLOG(X))
```

where *X* and *Y* are type COMPLEX.

CONDITION VALUES RETURNED

MTH$_INVARGMAT	Invalid argument in Math Library. Base is (0.,0.).
MTH$_FLOOVEMAT	Floating-point overflow in Math Library.
SS$_ROPRAND	Reserved operand.

EXAMPLES

1

```
C+
C       This FORTRAN example forms the result of raising a complex base
C       to a complex power using OTS$POWCC.
C
C       Declare Z1, Z2, Z3, and OTS$POWCC as complex values. Then OTS$POWCC
C       will return the complex result of z1**z2:    Z3 = OTS$POWCC(Z1,Z2),
C       where Z1, and Z2 are passed by value.
C-
           COMPLEX Z1,Z2,Z3,OTS$POWCC
C+
C       Generate a complex base.
C-
           Z1 = (2.0,3.0)
C+
C       Generate a complex power.
C-
           Z2 = (1.0,2.0)
C+
C       Compute the complex value of Z1**Z2.
C-
           Z3 = OTS$POWCC( %VAL(REAL(Z1)), %VAL(AIMAG(Z1)),
       +                   %VAL(REAL(Z2)), %VAL(AIMAG(Z2)))
           TYPE *, ' The value of',Z1,'**',Z2,' is',Z3
           END
```

This FORTRAN example uses OTS$POWCC to rasaise an F_floating complex base to an F_floating complex exponent.

The output generated by this program is as follows:

```
 The value of (2.000000,3.000000)** (1.000000,2.000000) is
(-0.4639565,-0.1995301)
```

2

```
C+
C     This FORTRAN example forms the result of raising a complex base
C     to a complex power using OTS$POWCGCG_R3.
C
C     Declare Z1, Z2, and Z3 as complex values. OTS$POWCGCG_R3
C     will return the complex result of z1**z2: Z3 = Z1**Z2.
C-
        COMPLEX*16 Z1,Z2,Z3
C+
C     Generate a complex base.
C-
        Z1 = (2.0,3.0)
C+
C     Generate a complex power.
C-
        Z2 = (1.0,2.0)
C+
C     Compute the complex value of Z1**Z2.
C-
        Z3 = Z1**Z2
        TYPE 1,Z1,Z2,Z3
    1   FORMAT(' The value of (',F11.8,',',F11.8,')**(',F11.8,
     +  ',',F11.8,') is (',F11.8,',',F11.8,').')
        END
```

This FORTRAN example program shows how to use OTS$POWCGCG_R3. Notice the high precision in the output generated by this program:

```
The value of ( 2.00000000, 3.00000000)**( 1.00000000, 2.00000000) is
(-0.46395650,-0.46395650).
```

OTS$POWCxJ—Raise a Complex Base to a Signed Longword Integer Exponent

These procedures return the complex result of raising a complex base to an integer exponent.

FORMAT

OTS$POWCJ *base ,exponent*
OTS$POWCDJ_R3 *base ,exponent*
OTS$POWCGJ_R3 *base ,exponent*

Each of the above three formats corresponds to one of the three floating-point complex types.

RETURNS

VMS Usage: **complex_number**
type: **F_floating complex**
access: **write only**
mechanism: **by value**

Complex result of raising a complex base to an integer exponent. OTS$POWCJ returns an F_floating complex number. OTS$POWCDJ_R3 returns a D_floating complex number. OTS$POWCGJ_R3 returns a G_floating complex number. In each format, the result and base are of the same data type.

ARGUMENTS

base

VMS Usage: **complex_number**
type: **F_floating complex, D_floating complex, G_floating complex**
access: **read only**
mechanism: **by value**

Complex base. The **base** argument contains the complex base. For OTS$POWCJ, **base** is an F_floating complex number. For OTS$POWCDJ_R3, **base** is a D_floating complex number. For OTS$POWCGJ_R3, **base** is a G_floating complex number.

exponent

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by value**

Exponent. The **exponent** argument is a signed longword integer containing the exponent.

DESCRIPTION

OTS$POWCJ, OTS$POWCDJ_R3, and OTS$POWCGJ_R3 return the complex result of raising a complex base to an integer exponent. The complex result is as follows:

Base	Exponent	Result
Any	Greater than 0	The product of (base*2^i), where *i* is each nonzero bit in **exponent**
(0.,0.)	Less than or equal to 0	Undefined exponentiation
Not (0.,0.)	Less than 0	The product of (base*2^i), where *i* is each nonzero bit in **exponent**
Not (0.,0.)	0	(1.0,0.0)

CONDITION VALUES SIGNALED

SS$_FLTDIV	Floating-point zero divide occurred.
SS$_FLTOVF	Floating-point overflow occurred.
MTH$_UNDEXP	Undefined exponentiation.

EXAMPLE

```
C+
C    This FORTRAN example forms the complex result of
C    raising a complex base to a NON-NEGATIVE integer
C    power using OTS$POWCJ.
C
C    Declare Z1, Z2, Z3, and OTS$POWCJ as complex values.
C    Then OTS$POWCJ will return the complex result of
C    Z1**Z2:   Z3 = OTS$POWCJ(Z1,Z2),
C    where Z1, and Z2 are passed by value.
C-
        COMPLEX Z1,Z3,OTS$POWCJ
        INTEGER Z2
C+
C    Generate a complex base.
C-
        Z1 = (2.0,3.0)
C+
C    Generate an integer power.
C-
        Z2 = 2
C+
C    Compute the complex value of Z1**Z2.
C-
        Z3 = OTS$POWCJ( %VAL(REAL(Z1)), %VAL(AIMAG(Z1)), %VAL(Z2))
        TYPE 1,Z1,Z2,Z3
  1     FORMAT(' The value of (',F10.8,',',F11.8,')**',I1,' is (',F11.8,
     +  ',',F12.8,').')
        END
```

The output generated by this FORTRAN program is as follows:

```
The value of (2.00000000, 3.00000000)**2 is (-5.00000000, 12.00000000).
```

OTS$POWDx—Exponentiation of D_floating Base

OTS$POWDD, OTS$POWDJ, and OTS$POWDR raise a D_floating base to a D_floating, longword, or F_floating exponent.

FORMAT

OTS$POWDD *base ,exponent*
OTS$POWDJ *base ,exponent*
OTS$POWDR *base ,exponent*

The above formats correspond to raising the D_floating base to a D_floating, longword, or F_floating exponent.

RETURNS

VMS Usage:	**floating_point**
type:	**D_floating**
access:	**write only**
mechanism:	**by value**

D_floating result. Regardless of the type of the exponent, the result is always a D_floating number.

ARGUMENTS

base

VMS Usage:	**floating_point**
type:	**D_floating**
access:	**read only**
mechanism:	**by value**

Base. The **base** argument is a D_floating number containing the base.

exponent

VMS Usage:	**floating_point**
type:	**D_floating, longword integer (signed), F_floating**
access:	**read only**
mechanism:	**by value**

Exponent. The **exponent** argument contains the exponent. For OTS$POWDD, **exponent** is a D_floating number. For OTS$POWDJ, **exponent** is a signed longword integer. For OTS$POWDR, **exponent** is an F_floating number.

DESCRIPTION

OTS$POWDD, OTS$POWDJ and OTS$POWDR raise a D_floating base to a D_floating, longword, or F_floating exponent.

OTS$POWDD, OTS$POWDJ, and OTS$POWDR use the same basic algorithm. However, the internal calculations and the floating-point result are computed at the same precision level as that of the base value.

OTS$POWDD

The D_floating result for OTS$POWDD is given by:

Base	Exponent	Result
=0	> 0	0.0
=0	=0	Undefined exponentiation
=0	<0	Undefined exponentiation
<0	Any	Undefined exponentiation
> 0	> 0	2[**exponent**•DLOG2(**base**)]
> 0	=0	1.0
> 0	<0	2[**exponent**•DLOG2(**base**)]

Floating-point overflow can occur.

Undefined exponentiation occurs if the base is 0 and the exponent is 0 or negative, or if the base is negative.

OTS$POWDJ

The floating-point result is as follows:

Base	Exponent	Result
Any	> 0	Product of (base•2^i) where *i* is each nonzero bit position in $\lvert$exponent$\rvert$
> 0	=0	1.0
=0	=0	Undefined exponentiation
<0	=0	1.0
> 0	<0	1.0/product of (base•2^i), where *i* is each nonzero bit position in $\lvert$exponent$\rvert$
=0	<0	Undefined exponentiation
<0	<0	1.0/product of (base•2^i) where *i* is each nonzero bit position in $\lvert$exponent$\rvert$

Floating-point overflow can occur.

Undefined exponentiation occurs if the base is 0 and the exponent is 0 or negative.

OTS$POWDR

OTS$POWDR converts the F_floating exponent to a D_floating number. OTS$POWDR then calculates the D_floating result in the same way that it calculates the result for OTS$POWDD.

CONDITION VALUES RETURNED

SS$_FLTOVF	Arithmetic trap. This error is signaled by the hardware if a floating-point overflow occurs.
MTH$_FLOOVEMAT	Floating-point overflow in Math Library.
MTH$_FLOUNDMAT	Floating-point underflow in Math Library.
MTH$_UNDEXP	Undefined exponentiation. This error is signaled if **base** is zero and **exponent** is zero or negative, or if the **base** is negative.

OTS$POWGx—Raise G_floating Base to G_floating or Longword Exponent

OTS$POWGG and OTS$POWGJ raise a G_floating base to a G_floating or longword exponent.

FORMAT

OTS$POWGG *base ,exponent*
OTS$POWGJ *base ,exponent*

RETURNS

VMS Usage: **floating_point**
type: **G_floating**
access: **write only**
mechanism: **by value**

Result of the exponentiation. Regardless of whether the base is raised to a G_floating or longword exponent, the result is always a G_floating number.

ARGUMENTS

base

VMS Usage: **floating_point**
type: **G_floating**
access: **read only**
mechanism: **by value**
Base which OTS$POWGx raises to a G_floating or longword exponent. The **base** argument is a G_floating number containing the base.

exponent

VMS Usage: **floating_point**
type: **G_floating, longword integer (signed)**
access: **read only**
mechanism: **by value**
Exponent to which OTS$POWGx raises the base. For OTS$POWGG, the **exponent** argument is a G_floating number containing the exponent. For OTS$POWGJ, the **exponent** argument is a signed longword integer containing the exponent.

DESCRIPTION

OTS$POWGG, and OTS$POWGJ raise a G_floating base to a G_floating or longword exponent.

OTS$POWGG and OTS$POWGJ use the same basic algorithm. However, the internal calculations and the floating-point result are computed at the same precision level as that of the base value.

OTS$POWGG

The G_floating result for OTS$POWGG is as follows:

Base	Exponent	Result
=0	> 0	0.0
=0	=0	Undefined exponentiation
=0	<0	Undefined exponentiation
<0	Any	Undefined exponentiation
> 0	> 0	2[**exponent**•GLOG2(**base**)]
> 0	=0	1.0
> 0	<0	2[**exponent**•GLOG2(**base**)]

Floating-point overflow can occur.

Undefined exponentiation occurs if the base is 0 and the exponent is 0 or negative, or if the base is negative.

OTS$POWGJ

The floating-point result is as follows:

Base	Exponent	Result
Any	> 0	Product of (base•2^i) where *i* is each nonzero bit position in |exponent|
> 0	=0	1.0
=0	=0	Undefined exponentiation
<0	=0	1.0
> 0	<0	1.0/product of (base•2^i), where *i* is each nonzero bit position in |exponent|
=0	<0	Undefined exponentiation
<0	<0	1.0/Product of (base•2^i) where *i* is each nonzero bit position in |exponent|

Floating-point overflow can occur.

Undefined exponentiation occurs if the base is 0 and the exponent is 0 or negative.

CONDITION VALUES RETURNED

SS$_FLTOVF	Arithmetic trap. This error is signaled by the hardware if a floating-point overflow occurs.
MTH$_FLOOVEMAT	Floating-point overflow in Math Library.
MTH$_FLOUNDMAT	Floating-point underflow in Math Library.
MTH$_UNDEXP	Undefined exponent. This error is signaled if **base** is zero and **exponent** is zero or negative, or if **base** is negative.

EXAMPLE

```
C+
C   This example demonstrates the use of OTS$POWGG,
C   which raises a G_floating point base
C   to a G_floating point power.
C-
        REAL*8 X,Y,RESULT,OTS$POWGG
C+
C   The arguments of OTS$POWGG are passed by value. FORTRAN can
C   only pass INTEGER and REAL*4 expressions as VALUE. Since
C   INTEGER and REAL*4 values are one longword long, while REAL*8
C   values are two longwords long, equate the base (and power) to
C   a two dimensional INTEGER vector. These vectors will be passed
C   by VALUE.
C-
        INTEGER N(2),M(2)
        EQUIVALENCE (N(1),X), (M(1),Y)
        X = 8.0
        Y = 2.0
C+
C  To pass X by value, pass N(1) and N(2) by value. Similarly for Y.
C-
        RESULT = OTS$POWGG(%VAL(N(1)),%VAL(N(2)),%VAL(M(1)),%VAL(M(2)))
        TYPE *,' 8.0**2.0 IS ',RESULT
        X = 9.0
        Y = -0.5
C+
C   In FORTRAN, OTS$POWWGG is indirectly called by simply using the
C   exponentiation operator.
C-
        RESULT = X**Y
        TYPE *,' 9.0**-0.5 IS ',RESULT
        END
```

This FORTRAN example uses OTS$POWGG to raise a G_floating base to a G_floating power.

The output generated by this example is as follows:

```
8.0**2.0 IS     64.0000000000000
8.0**2.0 IS     0.333333333333333
```

OTS$POWHx—Exponentiation, H_floating Base

OTS$POWHH_R3 and OTS$POWHJ_R3 raise an H_floating base to an H_floating or longword exponent.

FORMAT

OTS$POWHH_R3 *base ,exponent*
OTS$POWHJ_R3 *base ,exponent*

The above formats correspond to raising an H_floating number to either an H_floating or a signed longword integer exponent.

RETURNS

VMS Usage: **floating_point**
type: **H_floating**
access: **write only**
mechanism: **by value**

H_floating result. Regardless of the type of the exponent, the result is always an H_floating number.

ARGUMENTS

base

VMS Usage: **floating_point**
type: **H_floating**
access: **read only**
mechanism: **by value**
Base. The **base** argument is an H_floating number containing the base.

exponent

VMS Usage: **longword_signed**
type: **H_floating, longword integer (signed)**
access: **read only**
mechanism: **by value**
Exponent. The **exponent** argument contains the exponent. For OTS$POWHH_R3, **exponent** is an H_floating number. For OTS$POWHJ_R3, **exponent** is a signed longword integer.

DESCRIPTION

OTS$POWHH and OTS$POWHJ raise an H_floating base to an H_floating or longword exponent.

OTS$POWHH and OTS$POWHJ use the same basic algorithm. However, the internal calculations and the floating-point result are computed at the same precision level as that of the base value.

OTS$POWHH

The H_floating result for OTS$POWHH is as follows:

Base	Exponent	Result
= 0	> 0	0.0
= 0	= 0	Undefined exponentiation
= 0	< 0	Undefined exponentiation
< 0	Any	Undefined exponentiation
> 0	> 0	2[**exponent**•HLOG2(**base**)]
> 0	= 0	1.0
> 0	< 0	2[**exponent**•HLOG2(**base**)]

Floating-point overflow can occur.

Undefined exponentiation occurs if the base is 0 and the exponent is 0 or negative, or if the base is negative.

OTS$POWHJ

The floating-point result is as follows:

Base	Exponent	Result
Any	> 0	Product of (base•2^i) where *i* is each nonzero bit position in |exponent|
> 0	= 0	1.0
= 0	= 0	Undefined exponentiation
< 0	= 0	1.0
> 0	< 0	1.0/product of (base•2^i), where *i* is each nonzero bit position in |exponent|
= 0	< 0	Undefined exponentiation
< 0	< 0	1.0/product of (base•2^i) where *i* is each nonzero bit position in |exponent|

Floating-point overflow can occur.

Undefined exponentiation occurs if the base is 0 and the exponent is 0 or negative.

CONDITION VALUES RETURNED

SS$_FLTOVF	Arithmetic trap. This error is signaled by the hardware if a floating-point overflow occurs.
MTH$_FLOOVEMAT	Floating-point overflow in Math Library.
MTH$_FLOUNDMAT	Floating-point underflow in Math Library.
MTH$_UNDEXP	Undefined exponentiation. This error is signaled if **base** is zero and **exponent** is zero or negative, or if the **base** is negative.

EXAMPLE

```
C+
C Example of OTS$POWHH, which raises a H_floating
C point base to an H_floating point power. In FORTRAN,
C it is not directly called.
C-
        REAL*16 X,Y,RESULT
        X = 9877356535.0
        Y = -0.5837653
C+
C In FORTRAN, OTS$POWWHH is indirectly called by simply using the
C exponentiation operator.
C-
        RESULT = X**Y
        TYPE *,' 9877356535.0**-0.5837653 IS ',RESULT
        END
```

This FORTRAN example demonstrates how to call OTS$POWHH to raise an H_floating base to an H_floating power.

The output generated by this program is as follows:

```
8.0**2.0 IS    1.463779145994628357482343598205427E-0006
```

OTS$POWII—Exponentiation, Word Base

OTS$POWII raises a word base to a word exponent.

FORMAT

OTS$POWII *base ,exponent*

RETURNS

VMS Usage: **word_signed**
type: **word integer (signed)**
access: **write only**
mechanism: **by value**

Result of raising a word base to a word exponent.

ARGUMENTS

base

VMS Usage: **word_signed**
type: **word integer (signed)**
access: **read only**
mechanism: **by value**

Base. The **base** argument is a signed word integer containing the base.

exponent

VMS Usage: **word_signed**
type: **word integer (signed)**
access: **read only**
mechanism: **by value**

Exponent. The **exponent** argument is a signed word integer containing the exponent.

CONDITION VALUES RETURNED

SS$_FLTDIV	Arithmetic trap. This error is signaled by the hardware if a floating-point zero divide occurs.
SS$_FLTOVF	Arithmetic trap. This error is signaled by the hardware if a floating-point overflow occurs.
MTH$_UNDEXP	Undefined exponentiation. This error is signaled if **base** is zero and **exponent** is zero or negative, or if **base** is negative.

OTS$POWJJ—Raise a Longword Base to a Longword Exponent

OTS$POWJJ raises a signed longword base to a signed longword exponent.

FORMAT

OTS$POWJJ *base ,exponent*

RETURNS

VMS Usage:	**longword_signed**
type:	**longword integer (signed)**
access:	**write only**
mechanism:	**by value**

Result of raising a longword base to a longword exponent.

ARGUMENTS

base

VMS Usage:	**longword_signed**
type:	**longword integer (signed)**
access:	**read only**
mechanism:	**by value**

Base. The **base** argument is a signed longword integer containing the base.

exponent

VMS Usage:	**longword_signed**
type:	**longword integer (signed)**
access:	**read only**
mechanism:	**by value**

Exponent. The **exponent** argument is a signed longword integer containing the exponent.

CONDITION VALUES SIGNALED

SS$_FLTDIV	Arithmetic trap. This error is signaled by the hardware if a floating-point zero divide occurs.
SS$_FLTOVF	Arithmetic trap. This error is signaled by the hardware if a floating-point overflow occurs.
MTH$_UNDEXP	Undefined exponentiation. This error is signaled if **base** is zero and **exponent** is zero or negative, or if **base** is negative.

OTS$POWLULU
Raise an Unsigned Longword Integer Base to an Unsigned Longword Exponent

OTS$POWLULU returns the result of raising an unsigned longword integer base to an unsigned longword integer exponent.

FORMAT

OTS$POWLULU *base, exponent*

RETURNS

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Result of raising an unsigned longword integer base to an unsigned longword integer exponent. This function value is returned in R0.

ARGUMENTS

base

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Unsigned longword integer base. The **base** argument contains the value of the integer base.

exponent

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Unsigned longword integer exponent. The **exponent** argument contains the value of the integer exponent.

DESCRIPTION

OTS$POWLULU returns the unsigned longword integer result of raising an unsigned longword integer base to an unsigned longword integer exponent. Note that overflow cannot occur in this routine. If the result or intermediate result is greater than 32 bits, the low-order 32 bits are used.

CONDITION VALUES RETURNED

MTH$_UNDEXP	Both the base and exponent values are zero.

OTS$POWxLU
Raise a Floating-Point Base to an Unsigned Longword Integer Exponent

OTS$POWRLU, OTS$POWDLU, OTS$POWGLU, and OTS$POWHLU_R3 return the result of raising a floating-point base to an unsigned longword integer exponent.

FORMAT

OTS$POWRLU *base ,exponent*
OTS$POWDLU *base ,exponent*
OTS$POWGLU *base ,exponent*
OTS$POWHLU_R3 *base ,exponent*

RETURNS

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating, H_floating**
access: **write only**
mechanism: **by value**

Result of raising a floating-point base to an unsigned longword integer exponent. OTS$POWRLU returns an F_floating number. OTS$POWDLU returns a D_floating number. OTS$POWGLU returns a G_floating number. OTS$POWHLU_R3 returns an H_floating number.

ARGUMENTS

base

VMS Usage: **floating_point**
type: **F_floating, D_floating, G_floating, H_floating**
access: **read only**
mechanism: **by value**

Floating-point base. The **base** argument contains the value of the base. For OTS$POWRLU, **base** is an F_floating number. For OTS$POWDLU, **base** is a D_floating number. For OTS$POWGLU, **base** is a G_floating number. For OTS$POWHLU_R3, **base** is an H_floating number.

exponent

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Integer exponent. The **exponent** argument contains the value of the unsigned longword integer exponent.

DESCRIPTION

OTS$POWRLU, OTS$POWDLU, OTS$POWGLU, and OTS$POWHLU_R3 return the result of raising a floating-point base to an unsigned longword integer exponent. The floating-point result is as follows:

Base	Exponent	Result
Any	Greater than 0	The product of (base•2^i), where *i* is each nonzero bit in **exponent**
Greater than 0	0	1.0
0	0	Undefined exponentiation
Less than 0	0	1.0

CONDITION VALUES RETURNED

MTH$_FLOOVEMAT	Floating-point overflow in Math Library
MTH$_FLOUNDMAT	Floating-point underflow in Math Library. This can only occur if the caller has floating-point underflow enabled.
MTH$_UNDEXP	Undefined exponentiation. This occurs if both the **base** and **exponent** arguments are zero.

OTS$POWRx—Exponentiation, F_floating Base

OTS$POWRD, OTS$POWRJ, and OTS$POWRR raise an F_floating base to a D_floating, longword,or F_floating exponent.

FORMAT

OTS$POWRD *base ,exponent*
OTS$POWRJ *base ,exponent*
OTS$POWRR *base ,exponent*

The above formats correspond to raising the base to a D_floating, longword and F_floating exponent.

RETURNS

VMS Usage: **floating_point**
type: **F_floating**
access: **write only**
mechanism: **by value**

Result. OTS$POWRJ returns an F-floating number. OTS$POWRR returns an F_floating number. (If the exponent is F_floating or a longword integer, the result is F_floating.) OTS$POWRD returns a D_floating number.

ARGUMENTS

base

VMS Usage: **floating_point**
type: **F_floating**
access: **read only**
mechanism: **by value**
Base. The **base** argument is an F_floating number containing the base.

exponent

VMS Usage: **varying_arg**
type: **D_floating, longword integer (signed), F_floating**
access: **read only**
mechanism: **by value**
Exponent. The **exponent** argument contains the exponent. For OTS$POWRD, **exponent** is a D-floating number. For OTS$POWRJ, **exponent** is a signed longword integer. For OTS$POWRR, **exponent** is an F_floating number.

DESCRIPTION

OTS$POWRD, OTS$POWRJ and OTS$POWRR raise a F_floating base to a D_floating, longword, or F_floating exponent.

OTS$POWRD, OTS$POWRJ and OTS$POWRR use the same basic algorithm. However, the internal calculations and the floating-point result are computed at the same precision level as that of the base value.

OTS$POWRR

The F_floating result for OTS$POWRR is as follows:

Base	Exponent	Result
=0	>0	0.0
=0	=0	Undefined exponentiation
=0	<0	Undefined exponentiation
<0	Any	Undefined exponentiation
>0	>0	$2^{[exponent \cdot LOG2(base)]}$
>0	=0	1.0
>0	<0	$2^{[exponent \cdot LOG2(base)]}$

Floating-point overflow can occur.

Undefined exponentiation occurs if the base is zero and the exponent is zero or negative, or if the base is negative.

OTS$POWRJ

The floating-point result is as follows:

Base	Exponent	Result
Any	>0	Product of $(base \cdot 2^i)$ where *i* is each nonzero bit position in $\lvert exponent \rvert$
>0	=0	1.0
=0	=0	Undefined exponentiation
<0	=0	1.0
>0	<0	1.0/product of $(base \cdot 2^i)$, where *i* is each nonzero bit position in $\lvert exponent \rvert$
=0	<0	Undefined exponentiation
<0	<0	1.0/product of $(base \cdot 2^i)$ where *i* is each nonzero bit position in $\lvert exponent \rvert$

Floating-point overflow can occur.

Undefined exponentiation occurs if the base is 0 and the exponent is 0 or negative.

OTS$POWRD

OTS$POWRD first converts the F_floating base to D_floating. OTS$POWRD then calculates the F_floating result in the same way that it calculates the result for OTS$POWRR.

CONDITION VALUES RETURNED

SS$_FLTOVF	Arithmetic trap. This error is signaled by the hardware if a floating-point overflow occurs.
MTH$_FLOOVEMAT	Floating-point overflow in Math Library.
MTH$_FLOUNDMAT	Floating-point underflow in Math Library.
MTH$_UNDEXP	Undefined exponentiation. This error is signaled if **base** is zero and **exponent** is zero or negative, or if **base** is negative.

EXAMPLES

1

```
C+
C  This FORTRAN example demonstrates the use
C  of OTS$POWRR, which raises an F_floating
C  point base to an F_floating point power.
C-
        REAL*4 X,Y,RESULT,OTS$POWRR
        X = 8.0
        Y = 2.0
C+
C  The arguments of OTS$POWRR are passed by value.
C-
        RESULT = OTS$POWRR(%VAL(X),%VAL(Y))
        TYPE *,' 8.0**2.0 IS ',RESULT
        X = 9.0
        Y = -0.5
C+
C  In FORTRAN, OTS$POWRR is indirectly called by simply
C  using the exponentiation operator.
C-
        RESULT = X**Y
        TYPE *,' 9.0**-0.5 IS ',RESULT
        END
```

This FORTRAN example uses OTS$POWRR to raise an F_floating point base to an F_floating point power. The output generated by this program is as follows:

```
8.0**2.0 IS     64.00000
9.0**-0.5 IS    0.3333333
```

2

```
C+
C   This FORTRAN example demonstrates the use
C   of OTS$POWRD, which raises an F_floating point
C   base to a D_floating point power. The result is a
C   D_floating value.
C-
        REAL*4 X
        REAL*8 Y,RESULT,OTS$POWRD
        INTEGER M(2)
        EQUIVALENCE (M(1),Y)
        X = 9768.0
        Y = 9.0
C+
C  The arguments of OTS$POWRD are passed by value.
C-
        RESULT = OTS$POWRD(%VAL(X),%VAL(M(1)),%VAL(M(2)))
        TYPE *,' 9768.0**9.0 IS ',RESULT
        X = 7689.0
        Y = -0.587436654545
```

```
C+
C   In FORTRAN, OTS$POWRD is indirectly called by simply
C   using the exponentiation operator.
C-

        RESULT = X**Y
        TYPE *,' 7689.0**-0.587436654545 IS ',RESULT
        END
```

This FORTRAN example uses OTS$POWRD to raise an F_floating base to a D_floating exponent. Notice the difference in the precision of the result produced by this routine in comparison to the result produced by OTS$POWRR.

The output generated by this program is as follows:

```
9768.0**9.0 IS   8.095633864883290E+35
7689.0**-0.587436654545 IS   5.215519252836588E-03
```

OTS$SCOPY_DXDX—Copy a Source String Passed by Descriptor to a Destination String

OTS$SCOPY_DXDX copies a source string to a destination string. Both strings are passed by descriptor.

FORMAT

OTS$SCOPY_DXDX *src-str ,dst-str*

corresponding jsb entry point

OTS$SCOPY_DXDX6

RETURNS

VMS Usage:	**word_unsigned**
type:	**word (unsigned)**
access:	**write only**
mechanism:	**by value**

If **src-str** contains more characters than **dst-str**, and the JSB entry point is used, R0 contains the number of characters that were not copied.

ARGUMENTS

src-str

VMS Usage:	**char_string**
type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

Source string. The **src-str** argument is the address of a descriptor pointing to the source string. The descriptor class can be unspecified, fixed length, dynamic, scalar decimal, array, noncontiguous array, or varying.

dst-str

VMS Usage:	**char_string**
type:	**character string**
access:	**write only**
mechanism:	**by descriptor**

Destination string. The **dst-str** argument is the address of a descriptor pointing to the destination string. The class field determines the appropriate action.

See the Description section for further information.

DESCRIPTION

OTS$SCOPY_DXDX copies a source string to a destination string. All error conditions except truncation are signaled; truncation is ignored.

OTS$SCOPY_DXDX passes the source string by descriptor. In addition, an equivalent JSB entry point is provided, with R0 being the first argument (the descriptor of the source string), and R1 the second (the descriptor of the destination string).

For the CALL entry point, R0 (return status) is as it would be after a MOVC5 instruction. For the JSB entry point, R0:R5 and the PSL are as they would be after a MOVC5 instruction. R0:R5 contain the following:

R0 =	Number of bytes of source string not moved to destination string
R1 =	Address one byte beyond the last byte in the source string that was moved
R2 =	0
R3 =	Address one byte beyond the destination string
R4 =	0
R5 =	0

For further information, see the *VAX–11 Architecture Reference Manual.*

Depending on the class of the destination string, the actions described below occur:

Class Field	Action
DSC$K_CLASS_S,Z,SD,A,NCA	Copy the source string. If needed, space fill or truncate on the right.
DSC$K_CLASS_D	If the area specified by the destination descriptor is large enough to contain the source string, copy the source string and set the new length in the destination descriptor. If the area specified is not large enough, return the previous space allocation (if any) and then dynamically allocate the amount of space needed. Copy the source string and set the new length and address in the destination descriptor.
DSC$K_CLASS_VS	Copy source string to destination string up to the limit of DSC$W_MAXSTRLEN with no padding. Readjust current length field to actual number of bytes copied.

CONDITION VALUES SIGNALED

OTS$_FATINTERR	Fatal internal error.
OTS$_INVSTRDES	Invalid string descriptor.
OTS$_INSVIRMEM	Insufficient virtual memory.

OTS$SCOPY_R_DX—Copy a Source String Passed by Reference to a Destination String

OTS$SCOPY_R_DX copies a source string passed by reference to a destination string.

FORMAT

OTS$SCOPY_R_DX *src-len ,src-adr ,dst-str*

corresponding jsb entry point

OTS$SCOPY_R_DX6

RETURNS

VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by value**

If **src-str** contains more characters than **dst-str**, and the JSB entry point is used, R0 contains the number of characters that were not copied.

ARGUMENTS

src-len

VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **read only**
mechanism: **by value**

Length of the source string. The **src-len** argument is an unsigned word integer containing the length of the source string.

src-adr

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by reference**

Source string. The **src-adr** argument is the address of the source string.

dst-str

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

Destination string. The **dst-str** argument is the address of a descriptor pointing to the destination string. The class field determines the appropriate action. The length field (DSC$W_LENGTH) alone or both the address (DSC$A_POINTER) and length fields can be modified if the string is dynamic. For varying strings, the current length is rewritten.

DESCRIPTION

OTS$SCOPY_R_DX copies a source string to a destination string. All conditions except truncation are signaled; truncation is ignored. Input scalars are passed by value.

OTS$SCOPY_R_DX passes the source string by reference preceded by a length argument. In addition, an equivalent JSB entry point is provided, with R0 being the first argument, R1 the second, and R2 the third, if any. The length argument is passed in bits 15:0 of the appropriate register.

For the CALL entry point, R0 (return status) is as it would be after a MOVC5 instruction. For the JSB entry point, R0:R5 and the PSL are as they would be after a MOVC5 instruction. R0:R5 contain the following:

R0 =	Number of bytes of source string not moved to destination string
R1 =	Address one byte beyond the last byte in the source string that was moved
R2 =	0
R3 =	Address one byte beyond the destination string
R4 =	0
R5 =	0

(For additional information, see the *VAX–11 Architecture Reference Manual.*)

Depending on the class of the destination string, the actions described below occur:

Class Field	Action
DSC$K_CLASS_S,Z,SD,A,NCA	Copy the source string. If needed, space fill or truncate on the right.
DSC$K_CLASS_D	If the area specified by the destination descriptor is large enough to contain the source string, copy the source string and set the new length in the destination descriptor. If the area specified is not large enough, return the previous space allocation (if any) and then dynamically allocate the amount of space needed. Copy the source string and set the new length and address in the destination descriptor.
DSC$K_CLASS_VS	Copy source string to destination string up to the limit of DSC$W_MAXSTRLEN with no padding. Readjust current length field to actual number of bytes copied.

CONDITION VALUES SIGNALED

OTS$_FATINTERR	Fatal internal error.
OTS$_INVSTRDES	Invalid string descriptor.
OTS$_INSVIRMEM	Insufficient virtual memory.

EXAMPLE

A FORTRAN example demonstrating dynamic string manipulation appears at the end of OTS$SGET1_DD. This example uses OTS$SCOPY_R_DX, OTS$SGET1_DD, and OTS$SFREE1_DD.

OTS$SFREE1_DD—Strings, Free One Dynamic

OTS$SFREE1_DD returns one dynamic string area to free storage.

FORMAT

OTS$SFREE1_DD *dyn-dsc*

corresponding jsb entry point

OTS$SFREE1_DD6

RETURNS

None.

ARGUMENTS

dyn-dsc

VMS Usage: **quadword_unsigned**
type: **quadword (unsigned)**
access: **modify**
mechanism: **by reference**

Dynamic string descriptor. The **dyn-dsc** argument is the address of the dynamic string descriptor. The descriptor is assumed to be dynamic and its class field is not checked.

DESCRIPTION

OTS$SFREE1_DD deallocates the described string space and flags the descriptor as describing no string at all (DSC$A_POINTER = 0 and DSC$W_LENGTH = 0).

CONDITION VALUE SIGNALED

OTS$_FATINTERR	Fatal internal error.

EXAMPLE

A FORTRAN example demonstrating dynamic string manipulation appears at the end of OTS$SGET1_DD. This example uses OTS$SFREE1_DD, OTS$SGET1_DD, and OTS$SCOPY_R_DX.

OTS$SFREEN_DD—Strings, Free n Dynamic

OTS$SFREEN_DD takes as input a vector of one or more dynamic string areas and returns them to free storage.

FORMAT

OTS$SFREEN_DD *dsc-num ,first-dsc*

corresponding jsb entry point

OTS$SFREEN_DD6

RETURNS

None.

ARGUMENTS

dsc-num

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Number of adjacent descriptors to be flagged as having no allocated area (DSC$A_POINTER = 0 and DSC$W_LENGTH = 0) and to have their allocated areas returned to free storage by OTS$SFREEN_DD. The **dsc-num** argument is an unsigned longword containing this number.

first-dsc

VMS Usage: **quadword_unsigned**
type: **quadword (unsigned)**
access: **modify**
mechanism: **by reference**

First string descriptor of an array of string descriptors. The **first-dsc** argument is the address of the first string descriptor. The descriptors are assumed to be dynamic, and their class fields are not checked.

DESCRIPTION

OTS$SFREEN_DD6 deallocates the described string space and flags each descriptor as describing no string at all (DSC$A_POINTER = 0 and DSC$W_LENGTH = 0).

CONDITION VALUE SIGNALED

OTS$_FATINTERR	Fatal internal error.

OTS$SGET1_DD—Strings, Get One Dynamic

OTS$SGET1_DD allocates a specified number of bytes of dynamic virtual memory to a specified string descriptor.

FORMAT

OTS$SGET1_DD *len ,dyn-dsc*

corresponding jsb entry point **OTS$SGET1_DD_R6**

RETURNS

None.

ARGUMENTS

len

VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **read only**
mechanism: **by value**

Number of bytes to be allocated. The **len** argument contains the number of bytes. The amount of storage allocated is automatically rounded up. If the number of bytes is zero, a small number of bytes is allocated.

dyn-dsc

VMS Usage: **quadword_unsigned**
type: **quadword (unsigned)**
access: **modify**
mechanism: **by reference**

Dynamic string descriptor to which the area is to be allocated. The **dyn-str** argument is the address of the dynamic string descriptor. The class field is not checked but it is set to dynamic (DSC$B_CLASS = 2). The length field (DSC$W_LENGTH) is set to **len** and the address field (DSC$A_POINTER) is set to the string area allocated (first byte beyond the header).

DESCRIPTION

OTS$SGET1_DD allocates a specified number of bytes of dynamic virtual memory to a specified string descriptor. This procedure is identical to OTS$SCOPY_DXDX except that no source string is copied. You can write anything you want in the allocated area.

If the specified string descriptor already has dynamic memory allocated to it, but the amount allocated is either greater than or less than **len**, that space is deallocated before OTS$SGET1_DD allocates new space.

CONDITION VALUES SIGNALED

OTS$_FATINTERR	Fatal internal error.
OTS$_INSVIRMEM	Insufficient virtual memory.

EXAMPLE

```
      PROGRAM STRING_TEST
C+
C     This program demonstrates the use of some dynamic string
C     manipulation routines.
C-
C+
C     DECLARATIONS
C-
      IMPLICIT NONE
      CHARACTER*80    DATA_LINE
      INTEGER*4       DATA_LEN, DSC(2), CRLF_DSC(2), TEMP_DSC(2)
      CHARACTER*2     CRLF
C+
C     Initialize the output descriptor.  It should be empty.
C-
      CALL OTS$SGET1_DD(%VAL(0), DSC)
C+
C     Initialize a descriptor to the string CRLF and copy the
C     character CRLF to it.
C-
      CALL OTS$SGET1_DD(%VAL(2), CRLF_DSC)
      CRLF = CHAR(13)//CHAR(10)
      CALL OTS$SCOPY_R_DX( %VAL(2), %REF(CRLF(1:1)), CRLF_DSC)
C+
C     Initialize a temporary descriptor.
C-
      CALL OTS$SGET1_DD(%VAL(0), TEMP_DSC)
C+
C     Prompt the user.
C-
      WRITE(6, 999)
999   FORMAT(1X, 'Enter your message, end with CTRL/Z.')
C+
C     Read lines of text from the terminal until end-of-file.
C     Concatenate each line to the previous input.  Include a
C     CRLF between each line.
C-
      DO WHILE (.TRUE.)
          READ(5, 998, ERR = 10) DATA_LEN, DATA_LINE
998       FORMAT(Q,A)
          CALL OTS$SCOPY_R_DX( %VAL(DATA_LEN),
     1       %REF(DATA_LINE(1:1)),
     2       TEMP_DSC)
          CALL STR$CONCAT( DSC, DSC, TEMP_DSC, CRLF_DSC )
      END DO
C+
C     The user has typed CTRL/Z.  Output the data we read.
C-
10    CALL LIB$PUT_OUTPUT( DSC )
```

```
C+
C       Free the storage allocated to the dynamic strings.
C-
        CALL OTS$SFREE1_DD( DSC )
        CALL OTS$SFREE1_DD( CRLF_DSC )
        CALL OTS$SFREE1_DD( TEMP_DSC )
C+
C       End of program.
C-
        STOP
        END
```

This FORTRAN example program demonstrates dynamic string manipulation using OTS$SGET1_DD, OTS$SFREE1_DD, and OTS$SCOPY_R_DX.

SMG$ADD_KEY_DEF—Add Key Definition

SMG$ADD_KEY_DEF adds a keypad key definition to a table of key definitions.

FORMAT

SMG$ADD_KEY_DEF *key-table-id ,key-name [,if-state] [,attributes] [,equiv-string] [,state-string]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

key-table-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Identifies the key table to which you are adding a key definition. The **key-table-id** argument is the address of an unsigned longword that contains the key table identifier.

Key-table-id is returned by the SMG$CREATE_KEY_TABLE routine.

key-name

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Identifies the key whose value you are defining. The **key-name** argument is the address of a descriptor pointing to this key name. The SMG$ADD_KEY_DEF procedure changes the string to uppercase and removes trailing blanks.

Table 3-1 in Part I of this manual lists the valid key names.

if-state

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Qualifies the value returned when key-name is struck. The **if-state** argument is the address of a descriptor pointing to the state string.

If **if-state** is specified, this definition of **key-name** is used only if the current state matches the specified **if-state** string. The **if-state** argument must be from 1 to 31 characters in length. If this argument is omitted, **if-state** defaults to the value "DEFAULT."

attributes

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Longword bit mask specifying additional attributes of this key definition. The **attributes** argument is the address of an unsigned longword that contains this attribute mask. If omitted, the mask is zero.

Valid **attributes** are described in the following list:

SMG$M_KEY_NOECHO	If set, this bit specifies that **equiv_string** is not to be echoed when this key is pressed. If clear, **equiv_string** is echoed. If SMG$M_KEY_TERMINATE is not set, SMG$M_KEY_NOECHO is ignored.
SMG$M_KEY_TERMINATE	If set, this bit specifies that when this key is pressed (as qualified by **if-state**) that the input line is complete, and more characters should not be accepted. If clear, more characters may be accepted. In other words, setting this bit causes equivalence string to be treated as a terminator.
SMG$M_KEY_LOCKSTATE	If set, and if **state-string** is specified, the state name specified by **state-string** remains the current state until explicitly changed by a subsequent keystroke whose definition includes a **state-string**. If clear, the state name specified by **state-string** remains in effect only for the next defined keystroke.
SMG$M_KEY_PROTECTED	If set, this bit specifies that this key definition cannot be modified or deleted. If clear, the key definition can be modified or deleted.

The remaining bits are undefined and must be zero. It is possible to OR these values together to set more than one attribute at a time.

equiv-string

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Character string to be substituted for the keystroke in the returned line. The **equiv-string** argument is the address of a descriptor pointing to this equivalence string.

Equiv-string is echoed unless SMG$M_KEY_NOECHO is set. If **equiv-string** is omitted, no equivalence string is defined for this key.

state-string

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Contains a new state name which becomes the current state when this key is pressed. The **state-string** argument is the address of a descriptor pointing to the new state string.

If omitted, no new state is defined. If the current state is temporary (that is, if SMG$M_KEY_LOCKSTATE was not specified for the most recently pressed defined key), the current **state-string** becomes the null string.

DESCRIPTION

SMG$ADD_KEY_DEF inserts a key definition into a key definition table. The table must have been created with a call to SMG$CREATE_KEY_TABLE. After SMG$ADD_KEY_DEF executes, the specified equivalence string is returned when the user types the specified key.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_PREDEFREP	Successful completion. The previous key-definition has been replaced.
SMG$_INVDEFATT	Invalid key definition attributes.
SMG$_INVKTB_ID	Invalid **key-table-id**.
SMG$_KEYDEFPRO	Key definition is protected against change or deletion.
SMG$_WRONUMARG	Wrong number of arguments.

Any condition values returned by LIB$COPY_DXDX.

SMG$ALLOW_ESCAPE—Allow Escape Sequences

SMG$ALLOW_ESCAPE enables or disables SMG parsing of escape sequences which are output to a virtual display.

FORMAT

SMG$ALLOW_ESCAPE *display-id ,esc-flag*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Identifies the display in which output containing escape sequences is allowed. The **display-id** argument is the address of an unsigned longword that contains the display identifier. **Display-id** is returned by SMG$CREATE_VIRTUAL_DISPLAY.

esc-flag

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Determines whether escape sequence parsing is enabled or disabled. The **esc-flag** argument is the address of an unsigned longword that contains the escape flag. If **esc-flag** equals 1, parsing is on; if **esc-flag** equals 0, parsing is off.

DESCRIPTION

Normally, text written to a virtual display cannot contain escape sequences. SMG$ALLOW_ESCAPE lets escape sequences be interpreted as valid operations (for example, as a video attribute or a cursor positioning command). Thus, if **esc-flag** equals 1, the Screen Management Facility attempts to interpret the escape sequence and call the Screen Management routine that performs that function. If **esc-flag** equals 0, an error occurs if an escape sequence is detected. Note, however, that the virtual display must be pasted.

Note: **SMG$ALLOW_ESCAPE is intended to allow existing programs with embedded VT52/VT100 escape sequences to be converted to SMG$. SMG$ALLOW_ESCAPE should not be used in new program development.**

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_INVDIS_ID	Invalid **display-id**.
SMG$_INVARG	Invalid argument.

SMG$BEGIN_DISPLAY_UPDATE Begin Batching of Display Updates

SMG$BEGIN_DISPLAY_UPDATE saves, or batches, all output to a virtual display until a matching call to SMG$END_DISPLAY_UPDATE is encountered.

FORMAT

SMG$BEGIN_DISPLAY_UPDATE *display-id*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENT

display-id
VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the virtual display for which output is to be batched. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

Display-id is returned by SMG$CREATE_VIRTUAL_DISPLAY.

DESCRIPTION

SMG$BEGIN_DISPLAY_UPDATE lets you make more than one change to a display and have the changes appear only after all changes are complete. Thus, the user sees the display change from its initial state to its final state, without seeing any of the intermediate states.

Batching terminates when SMG$END_DISPLAY_UPDATE has been called the same number of times as has SMG$BEGIN_DISPLAY_UPDATE for a given display. The Screen Management Facility keeps track of batching for a given display; thus, the calls to the SMG$BEGIN_DISPLAY_UPDATE and SMG$END_DISPLAY_UPDATE need not occur in the same module.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_BATWAS_ON	Successful completion; note that batching had already been initiated.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_INVDIS_ID	Invalid **display-id**.

SMG$BEGIN_PASTEBOARD_UPDATE Begin Batching of Pasteboard Updates

SMG$BEGIN_PASTEBOARD_UPDATE saves, or batches, all output to a pasteboard until a matching call to SMG$END_PASTEBOARD_UPDATE is encountered.

FORMAT

SMG$BEGIN_PASTEBOARD_UPDATE *pasteboard-id*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENT

pasteboard-id

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Specifies the pasteboard for which output is to be batched. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier. **Pasteboard-id** is returned by SMG$CREATE_PASTEBOARD.

DESCRIPTION

SMG$BEGIN_PASTEBOARD_UPDATE lets you make more than one change to a pasteboard and have the changes appear only after all changes are complete. Thus, the user sees the pasteboard change from its initial state to its final state, without seeing any of the intermediate states.

Batching terminates when SMG$END_PASTEBOARD_UPDATE has been called the same number of times as has SMG$BEGIN_PASTEBOARD_UPDATE for a given pasteboard. The Screen Management Facility keeps track of batching for a given pasteboard; thus, the calls to the SMG$BEGIN_PASTEBOARD_UPDATE and SMG$END_PASTEBOARD_UPDATE need not occur in the same module.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_BATWAS_ON	Successful completion; note that batching had already been initiated.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_INVPAS_ID	Invalid **pasteboard-id**.

SMG$CANCEL_INPUT—Cancel Input Request

SMG$CANCEL_INPUT immediately cancels any read-in-progress that was issued by SMG$READ_COMPOSED_LINE, SMG$READ_KEYSTROKE, SMG$READ_STRING or SMG$READ_VERIFY.

FORMAT

SMG$CANCEL_INPUT *keyboard-id*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENT

keyboard-id

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Specifies the virtual keyboard for which the input is to be cancelled. The **keyboard-id** argument is the address of an unsigned longword that contains the keyboard identifier. **Keyboard-id** is returned by SMG$CREATE_VIRTUAL_KEYBOARD.

DESCRIPTION

SMG$CANCEL_INPUT causes immediate termination of an SMG$READ_COMPOSED_LINE, SMG$READ_KEYSTROKE, SMG$READ_STRING or SMG$READ_VERIFY input operation from a terminal. The condition code SS$_ABORT is returned to those routines when you use SMG$CANCEL_INPUT. Note that if the specified virtual keyboard is associated with an RMS file, this procedure has no effect because it is not possible to cancel an outstanding RMS input operation.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_RMSNOTCAN	Successful completion. RMS operation cannot be cancelled.
SMG$_INVKBD_ID	Invalid **keyboard-id**.
SMG$_WRONUMARG	Wrong number of arguments.

SMG$CHANGE_PBD_CHARACTERISTICS Change Pasteboard Characteristics

SMG$CHANGE_PBD_CHARACTERISTICS lets you change the width, height, and background color associated with a pasteboard.

FORMAT

SMG$CHANGE_PBD_CHARACTERISTICS *pasteboard-id [,desired-width] [,resulting-width] [,desired-height] [,resulting-height] [,desired-background-color] [,resulting-background-color]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

pasteboard-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the pasteboard whose characteristics are to be changed. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

Pasteboard-id is returned by SMG$CREATE_PASTEBOARD.

desired-width

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

New width for the pasteboard. The **desired-width** argument is the address of a signed longword integer that contains the desired width. If omitted, the width does not change.

resulting-width

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **write only**
mechanism: **by reference**

Physical width of the pasteboard. The **resulting-width** argument is the address of a signed longword integer into which is written the actual width of the pasteboard.

Resulting-width may be larger than **desired-width** if the terminal cannot be set exactly to the **desired-width**. **Resulting-width** may be smaller than **desired-width** if the physical width of the terminal is smaller than **desired-width**.

desired-height

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

New height for the pasteboard. The **desired-height** argument is the address of a signed longword integer that contains the desired height of the pasteboard. If omitted, the height does not change.

resulting-height

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **write only**
mechanism: **by reference**

Physical height of the pasteboard. The **resulting-height** argument is the address of a signed longword integer into which is written the actual height of the pasteboard.

Resulting-height may be larger than **desired-height** if the terminal cannot be set exactly to the **desired-height**. **Resulting-height** may be smaller than **desired-height** if the physical height of the terminal is smaller than **desired-height**.

desired-background-color

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Symbolic name for the desired color. The **desired-background-color** argument is the address of an unsigned longword that contains the desired color.

If omitted, the background color is not changed. Valid choices are SMG$C_COLOR_WHITE and SMG$C_COLOR_BLACK.

resulting-background-color

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Receives the actual color chosen. The **resulting-background-color** argument is the address of an unsigned longword into which is written the actual background color. If the terminal does not support the specified color, the nearest approximation is chosen.

This routine may return SMGC_COLOR_WHITE, SMGC_COLOR_BLACK, or SMG$C_COLOR_UNKNOWN. If **desired-background-color** is omitted, the value of **resulting-background-color** does not change.

DESCRIPTION

SMG$CHANGE_PBD_CHARACTERISTICS lets you change the width, height, and background color associated with a pasteboard.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_PBDIN_USE	Cannot change characteristics while batching is on.
SMG$_INVWIDARG	Invalid width of 0 desired.
SMG$_INVPAGARG	Invalid height of 0 desired.
SMG$_INVCOLARG	Unknown background color specified.

SMG$CHANGE_RENDITION Change Default Rendition

SMG$CHANGE_RENDITION changes the video attributes for all or part of a virtual display.

FORMAT

SMG$CHANGE_RENDITION *display-id ,start-row ,start-col ,rows ,columns [,rendition-set] [,rendition-complement]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the virtual display whose default rendition is to be changed. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

Display-id is returned by SMG$CREATE_VIRTUAL_DISPLAY.

start-row

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Starting row position to receive the new rendition. The **start-row** argument is the address of a signed longword integer that contains the number of the starting row.

start-col

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Starting column position to receive the new rendition. The **start-col** argument is the address of a signed longword integer that contains the number of the starting column.

rows

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Number of rows to receive the new rendition. The **rows** argument is the address of a signed longword integer that contains the number of rows to be affected.

columns

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Number of columns to receive the new rendition. The **columns** argument is the address of a signed longword integer that contains the number of columns to be affected.

rendition-set

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Mask which denotes video attributes for the specified portion of the virtual display. The **rendition-set** argument is the address of an unsigned longword whose bits control the video rendition. Each bit in this argument affects the corresponding attribute in the display.

Video attributes which can be manipulated in this manner are as follows:

SMG$M_BLINK	Displays blinking characters
SMG$M_BOLD	Displays characters in higher-than-normal intensity
SMG$M_REVERSE	Displays characters in reverse video, that is, using the opposite default rendition of the virtual display
SMG$M_UNDERLINE	Displays underlined characters

The **rendition-set/rendition-complement** scheme works as follows:

- Each display has a default rendition associated with it when it is created.
- When SMG$CHANGE_RENDITION is called, each bit in the default rendition set is ORed (the Boolean OR operation) with the corresponding bit in the **rendition-set** argument.
- The result of this operation is then XORed (the Boolean EXCLUSIVE OR operation) with each bit in the **rendition-complement** argument.

Using these two arguments, the caller can exercise independent control over each attribute in a single call.

rendition-complement

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Mask which denotes video attributes for the specified portion of the virtual display. The **rendition-complement** argument is the address of an unsigned longword whose bits control the video rendition.

Each bit in **rendition-complement** affects the corresponding attribute in the display. Video attributes which can be manipulated in this manner are the same as for the **rendition-set** argument.

The following table shows the action taken by the Screen Management Facility for various combinations of **rendition-set** and **rendition-complement** attributes:

Set	Complement	Action
0	0	Attribute unchanged
1	0	Attribute on
0	1	Attribute set to complement of default setting
1	1	Attribute off

DESCRIPTION

This procedure changes the default video rendition of a rectangular block of text already in the specified virtual display. For example, you might use this procedure to redisplay a particular row in reverse video.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_NO_CHADIS	Successful completion. No change to virtual display.
SMG$_INVROW	Invalid start row. The specified row is outside the virtual display.
SMG$_INVCOL	Invalid start column. The specified column is outside the virtual display.
SMG$_INVDIS_ID	Invalid **display-id**.
SMG$_INVARG	Invalid number of rows, invalid number of columns, unrecognized **rendition-set** code, or unrecognized **rendition-complement** code.
SMG$_WRONUMARG	Wrong number of arguments.

SMG$CHANGE_VIRTUAL_DISPLAY Change Virtual Display

SMG$CHANGE_VIRTUAL_DISPLAY lets you change the dimensions, border, and video attributes of a virtual display.

FORMAT

SMG$CHANGE_VIRTUAL_DISPLAY *display-id ,rows ,columns [,display-attributes] [,video-attributes] [,char-set]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the virtual display whose attributes are to be changed. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

Display-id is returned by SMG$CREATE_VIRTUAL_DISPLAY.

rows

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the new number of rows for the virtual display. The **rows** argument is the address of a signed longword integer that contains the number of rows in the virtual display.

columns

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the new number of columns for the virtual display. The **columns** argument is the address of a signed longword integer that contains the number of columns in the virtual display.

display-attributes

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies whether the virtual display is bordered (the default). The **display-attributes** argument is the address of an unsigned longword that contains the display attributes mask.

To explicitly specify a bordered display, use the mask SMG$M_BORDER. All other bits are reserved for use by DIGITAL and must be zero.

video-attributes

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the default rendition to be applied to all output in a virtual display, unless overridden by a call to a specific output routine. The **video-attributes** argument is the address of an unsigned longword that contains the video attributes mask.

For example, a call to SMG$PUT_CHARS with an explicit rendition specified would override the default rendition.

The bits that can be set for this argument are as follows:

SMG$M_BLINK	Sets the display default to blinking characters
SMG$M_BOLD	Sets the display default to characters in higher-than-normal intensity
SMG$M_REVERSE	Sets the display default to characters in reverse video, that is, to the opposite of the current default rendition of the virtual display
SMG$M_UNDERLINE	Sets the display default to underlined characters

Note that you can specify any combination of attributes in a single call. All other bits are reserved for use by DIGITAL and must be zero.

char-set

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the default character set for all text in this virtual display. The **char-set** argument is the address of an unsigned longword that contains the character set specifier. At this time, the only valid value is SMG$C_ASCII, which is also the default.

DESCRIPTION

SMG$CHANGE_VIRTUAL_DISPLAY lets you change the size or default attributes of an existing virtual display. If the size of the virtual display is changed, the Screen Management Facility attempts to remap the text associated with the display to fit the new dimensions (starting at row and column 1). If the new size of the virtual display is smaller than the old size, text may be truncated. If the new size of the virtual display is larger than the old size, text may be padded on the right with spaces.

When a display is redimensioned, the virtual cursor for the display is moved to row 1 and column 1. Note that if a labeled border applies to the virtual display and does not fit the redimensioned display, the label is deleted.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
LIB$_INSVIRMEM	Insufficient virtual memory to reallocate needed buffers.
SMG$_INVARG	Invalid video or display attributes.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_INVDIS_ID	Invalid **display-id**.

SMG$CHECK_FOR_OCCLUSION Check For Occlusion

SMG$CHECK_FOR_OCCLUSION checks to see whether a virtual display is covered by another virtual display.

FORMAT

SMG$CHECK_FOR_OCCLUSION *display-id ,pasteboard-id ,occlusion-state*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

display-id

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Specifies the virtual display to be checked. The **display-id** argument is the address of an unsigned longword that contains the display identifier. **Display-id** is returned by SMG$CREATE_VIRTUAL_DISPLAY.

pasteboard-id

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Specifies the pasteboard to be checked. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier. **Pasteboard-id** is returned by SMG$CREATE_PASTEBOARD.

occlusion-state

VMS Usage:	**longword_signed**
type:	**longword integer (signed)**
access:	**write only**
mechanism:	**by reference**

Receives the value denoting whether the display is occluded. The **occlusion-state** argument is the address of a signed longword integer into which the occlusion state is written. **Occlusion-state** is set to 1 if the display is occluded or set to 0 if the display is not occluded on the specified pasteboard. If the procedure does not return SS$_NORMAL, the contents of **occlusion-state** are undefined.

DESCRIPTION

SMG$CHECK_FOR_OCCLUSION determines whether a specified virtual display (as pasted to the specified pasteboard) is occluded, or covered up, by another virtual display.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_NOTPASTED	Specified virtual display is not pasted to the specified pasteboard.
SMG$_INVPAS_ID	Invalid **pasteboard-id**.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_INVDIS_ID	Invalid **display-id**.

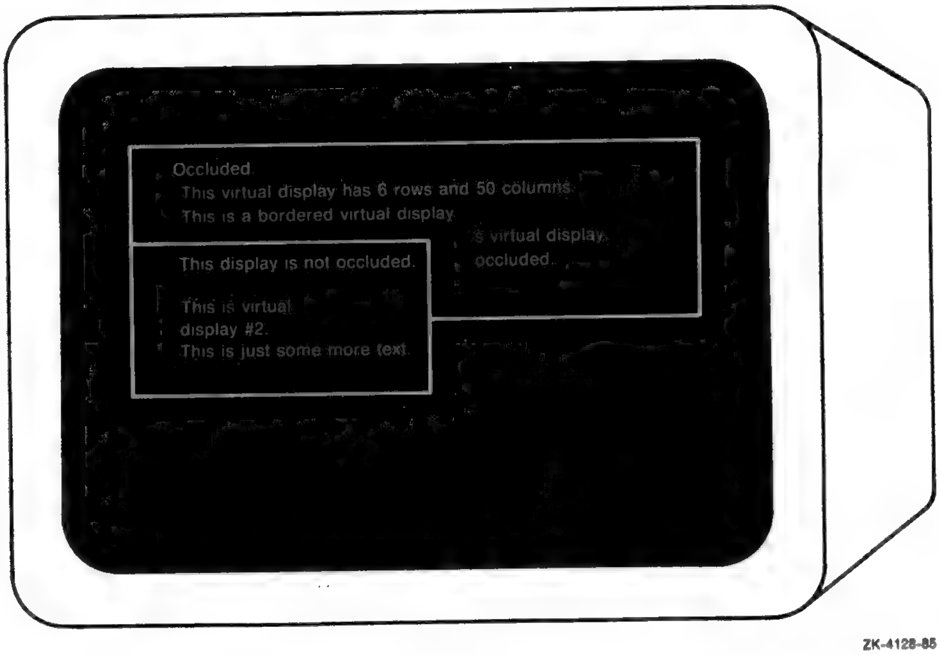
EXAMPLE

```
C+
C This FORTRAN example program demonstrates the use of
C SMG$CHECK_FOR_OCCLUSION.
C
C This routine creates a virtual display and writes it to the
C pasteboard.  Data is placed in the virtual display via SMG$PUT_CHARS.
C-
        INTEGER SMG$CREATE_VIRTUAL_DISPLAY, SMG$CREATE_PASTEBOARD,
        INTEGER SMG$PASTE_VIRTUAL_DISPLAY,  SMG$PUT_CHARS
        INTEGER SMG$CHECK_FOR_OCCLUSION
        INTEGER DISPLAY1, DISPLAY2, PASTE1, PASTE2, ROWS, COLUMNS, BORDER
        INTEGER OCCLUSION, STATUS
        CHARACTER*29 TEXT

C+
C Include the SMG definitions. In particular, we want SMG$M_BORDER.
C-
        INCLUDE '($SMGDEF)'

C+
C Create two virtual displays using SMG$CREATE_VIRTUAL_DISPLAY.
C Give them borders.
C-
        ROWS = 6
        COLUMNS = 50
        STATUS = SMG$CREATE_VIRTUAL_DISPLAY
     1          (ROWS, COLUMNS, DISPLAY1, SMG$M_BORDER)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        ROWS = 5
        COLUMNS = 30
        STATUS = SMG$CREATE_VIRTUAL_DISPLAY
     1          (ROWS, COLUMNS, DISPLAY2, SMG$M_BORDER)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Create the pasteboard using SMG$CREATE_PASTEBOARD.
C-
        STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        STATUS = SMG$CREATE_PASTEBOARD (PASTE2)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Use SMG$PUT_CHARS to put data into the virtual displays.
C-
        STATUS = SMG$PUT_CHARS ( DISPLAY1,
     1          ' This virtual display has 6 rows and 50 columns.', 2, 1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
```

```
        STATUS = SMG$PUT_CHARS ( DISPLAY1,
     1         ' This is a bordered virtual display.', 3, 1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        STATUS = SMG$PUT_CHARS ( DISPLAY1,
     1         ' SMG$PUT_CHARS puts data in this virtual display.', 4,
     1         1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        STATUS = SMG$PUT_CHARS ( DISPLAY1,
     1         ' This text should be partially occulded.', 5, 1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        STATUS = SMG$PUT_CHARS ( DISPLAY1,
     1         ' So should part of this row.', 6, 1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        STATUS = SMG$PUT_CHARS ( DISPLAY2, ' This is virtual', 3, 1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        STATUS = SMG$PUT_CHARS ( DISPLAY2,
     1         ' display #2.', 4, 1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        STATUS = SMG$PUT_CHARS ( DISPLAY2,
     1         ' This is just some more text.', 5, 1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Use SMG$PASTE_VIRTUAL_DISPLAY to paste the virtual display.
C-
        STATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DISPLAY1, PASTE1, 4, 15)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        STATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DISPLAY2, PASTE2, 8, 15)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Check the two virtual displays for occlusion by calling
C SMG$CHECK_FOR_OCCLUSION.
C-
        TEXT = 'This display is not occluded.'
        STATUS = SMG$CHECK_FOR_OCCLUSION (DISPLAY1, PASTE1, OCCLUSION)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        IF (OCCLUSION .EQ. 0) THEN
                STATUS = SMG$PUT_CHARS (DISPLAY1, TEXT, 1, 1)
                IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        ELSE
                STATUS = SMG$PUT_CHARS (DISPLAY1, 'Occluded.', 1 , 1)
                IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        END IF
        STATUS = SMG$CHECK_FOR_OCCLUSION (DISPLAY2, PASTE2, OCCLUSION)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        IF (OCCLUSION .EQ. 0) THEN
                STATUS = SMG$PUT_CHARS (DISPLAY2, TEXT, 1, 1)
                IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        ELSE
                STATUS = SMG$PUT_CHARS (DISPLAY2, 'Occluded.', 1 , 1)
                IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        END IF
        END
```

The output generated by this FORTRAN program is shown in Figure RTL-9.

Figure RTL-9 Output generated by FORTRAN Program Calling SMG$CHECK_FOR_OCCLUSION



SMG$CONTROL_MODE—Control Mode

SMG$CONTROL_MODE controls the mode of the pasteboard. This includes buffering, minimal updating, whether the screen is cleared when the pasteboard is deleted, and whether tab characters are used for screen formatting.

FORMAT

SMG$CONTROL_MODE *pasteboard-id [,new-mode] [,old-mode]*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

pasteboard-id

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Specifies the pasteboard to be changed. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier. **Pasteboard-id** is returned by SMG$CREATE_PASTEBOARD.

new-mode

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Specifies the new control settings to be used. The **new-mode** argument is the address of an unsigned longword that contains the mode settings. A bit set to 1 forces that mode to be employed while a bit set to 0 inhibits that mode of operation.

Valid settings are as follows:

SMG$M_BUF_ENABLED	Enables buffering.
SMG$M_MINUPD	Enables minimal update.
SMG$M_CLEAR_SCREEN	Causes the Screen Management Facility to clear the screen when the program exits if you have not previously deleted the pasteboard.
SMG$M_NOTABS	Causes the Screen Management Facility not to use tab characters to format the screen.

All other bits must be zero and are reserved for future use by DIGITAL.

old-mode

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Receives the control settings that were in effect prior to calling this procedure. The **old-mode** argument is the address of an unsigned longword into which is written the former mode settings. A bit set to 1 indicates that the specified mode was employed while a bit set to 0 indicates that the mode was inhibited.

DESCRIPTION

SMG$CONTROL_MODE lets you interrogate and change the modes of the Screen Management Facility operation for a specified pasteboard. This routine has two optional parameters, **new-mode** and **old-mode**. By specifying different combinations of these arguments, SMG$CONTROL_MODE can be used in various ways.

- To use SMG$CONTROL_MODE to determine the current mode settings, use the following format:

  SMG$CONTROL_MODE (pasteboard_id ,,old_mode)

- To use SMG$CONTROL_MODE to set the bits without regard to their current setting, use the following format:

  SMG$CONTROL_MODE (pasteboard_id ,new_mode)

- To use SMG$CONTROL_MODE to save the current settings, set new modes, and later restore the original settings, use the following format:

  SMG$CONTROL_MODE (pasteboard_id ,new_mode ,save_old_settings)

  This retrieves the current bit settings and then sets the mode according to the **new-mode** argument.

  Later, to restore the mode to its former state, specify the following format:

  SMG$CONTROL_MODE (pasteboard_id ,save_old_settings)

  This sets the new mode setting according to those previously retrieved.

Note that if both arguments are omitted, no information is returned.

The modes that can be interrogated and changed using SMG$CONTROL_MODE are as follows:

Buffering

In this mode, the Screen Management Facility buffers all output for efficient use of system QIOs. By calling SMG$FLUSH_BUFFER, the user can force to the screen any output that has been placed in the pasteboard buffer but not yet written to the terminal.

Minimal Screen Update

By default, the Screen Management Facility tries to minimize the number of characters actually sent to the terminal. It does this by sending only those characters that have changed.

Nonminimal updating rewrites any line containing a change, starting with the first changed character on that line.

Clear Screen

By default, the Screen Management Facility clears the screen when the program exits if you have not already deleted the pasteboard. To prevent this default behavior, turn the clear screen bit off.

No Tabs

If this bit is set, the Screen Management Facility does not rely on the terminal's tab settings. If it is not set, the Screen Management Facility will use physical tabs for the minimal update procedure. However, note that such use implicitly assumes that the tab stops are set to the DIGITAL default locations. Specify "no tabs" if you want to be sure that the application will run regardless of the tab settings the user has set on the terminal. By default, this bit is clear.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVARG	Invalid argument. **New-mode** has a bit set which does not correspond to SMG$M_BUF_ENABLED, SMG$M_MINUPD, SMG$M_CLEAR_SCREEN, or SMG$M_NOTABS.
SMG$_INVPAS_ID	Invalid **pasteboard-id**.
SMG$_WRONUMARG	Wrong number of arguments.

SMG$COPY_VIRTUAL_DISPLAY—Copy a Virtual Display

SMG$COPY_VIRTUAL_DISPLAY creates a copy of an existing virtual display and assigns to it a new virtual display number.

FORMAT

SMG$COPY_VIRTUAL_DISPLAY *curr-display-id ,new-display-id*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

curr-display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display identifier of the virtual display to be replicated. The **curr-display-id** argument is the address of the unsigned longword that contains the display identifier.

new-display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Receives the display identifier of the newly created virtual display. The **new-display-id** argument is the address of the unsigned longword that receives the new display identifier.

DESCRIPTION

SMG$COPY_VIRTUAL_DISPLAY creates a copy of an existing virtual display and assigns to it a new virtual display number. This newly created virtual display will not be pasted anywhere; use SMG$PASTE_VIRTUAL_DISPLAY and the **new-display-id** identifier to paste the newly created virtual display. The existing display being replicated does not have to be pasted when SMG$COPY_VIRTUAL_DISPLAY is invoked.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
LIB$_INSVIRMEM	Insufficient virtual memory to allocate needed buffer.

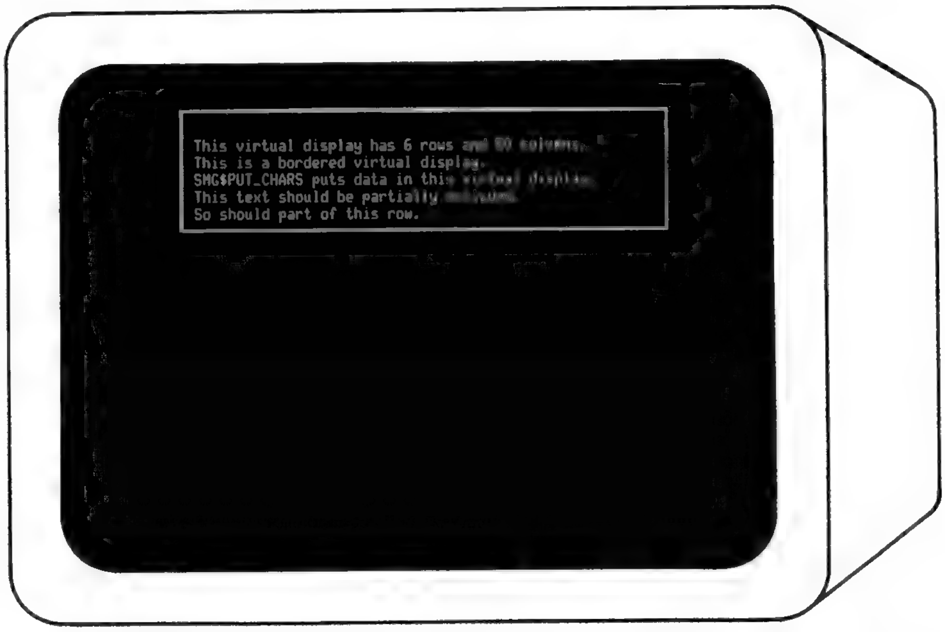
EXAMPLE

```
C+
C This FORTRAN example program demonstrates the use of
C SMG$COPY_VIRTUAL_DISPLAY.
C
C This routine creates a virtual display and writes it to the
C pasteboard.  Data is placed in the virtual display via SMG$PUT_CHARS.
C-
        IMPLICIT INTEGER (A-Z)
        CHARACTER*29 TEXT
C+
C Include the SMG definitions. In particular, we want SMG$M_BORDER.
C-
        INCLUDE '($SMGDEF)'
C+
C Create two virtual displays using SMG$CREATE_VIRTUAL_DISPLAY.
C Give them borders.
C-
        ROWS = 6
        COLUMNS = 50
        STATUS = SMG$CREATE_VIRTUAL_DISPLAY
     1          (ROWS, COLUMNS, DISPLAY1, SMG$M_BORDER)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        ROWS = 5
        COLUMNS = 30
        STATUS = SMG$CREATE_VIRTUAL_DISPLAY
     1          (ROWS, COLUMNS, DISPLAY2, SMG$M_BORDER)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Create the pasteboard using SMG$CREATE_PASTEBOARD.
C-
        STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        STATUS = SMG$CREATE_PASTEBOARD (PASTE2)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Use SMG$PUT_CHARS to put data into the virtual displays.
C-
        STATUS = SMG$PUT_CHARS ( DISPLAY1,
     1          ' This virtual display has 6 rows and 50 columns.', 2, 1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        STATUS = SMG$PUT_CHARS ( DISPLAY1,
     1          ' This is a bordered virtual display.', 3, 1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        STATUS = SMG$PUT_CHARS ( DISPLAY1,
     1          ' SMG$PUT_CHARS puts data in this virtual display.', 4,
     1          1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        STATUS = SMG$PUT_CHARS ( DISPLAY1,
     1          ' This text should be partially occluded.', 5, 1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        STATUS = SMG$PUT_CHARS ( DISPLAY1,
     1          ' So should part of this row.', 6, 1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        STATUS = SMG$PUT_CHARS ( DISPLAY2, ' This is virtual', 3, 1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        STATUS = SMG$PUT_CHARS ( DISPLAY2,
     1          ' display #2.', 4, 1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        STATUS = SMG$PUT_CHARS ( DISPLAY2,
     1          ' This is just some more text.', 5, 1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
```

```
C+
C Use SMG$PASTE_VIRTUAL_DISPLAY to paste the virtual display.
C-
        STATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DISPLAY1, PASTE1, 4, 15)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        STATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DISPLAY2, PASTE2, 8, 15)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Copy the first virtual display, the one that is partially occluded.
C-
STATUS = SMG$COPY_VIRTUAL_DISPLAY ( DISPLAY1, NEW_DISPLAY)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Now paste this new virtual display so that it occludes the other displays.
C-
        STATUS = SMG$PASTE_VIRTUAL_DISPLAY ( NEW_DISPLAY, PASTE1, 4, 20)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        END
```

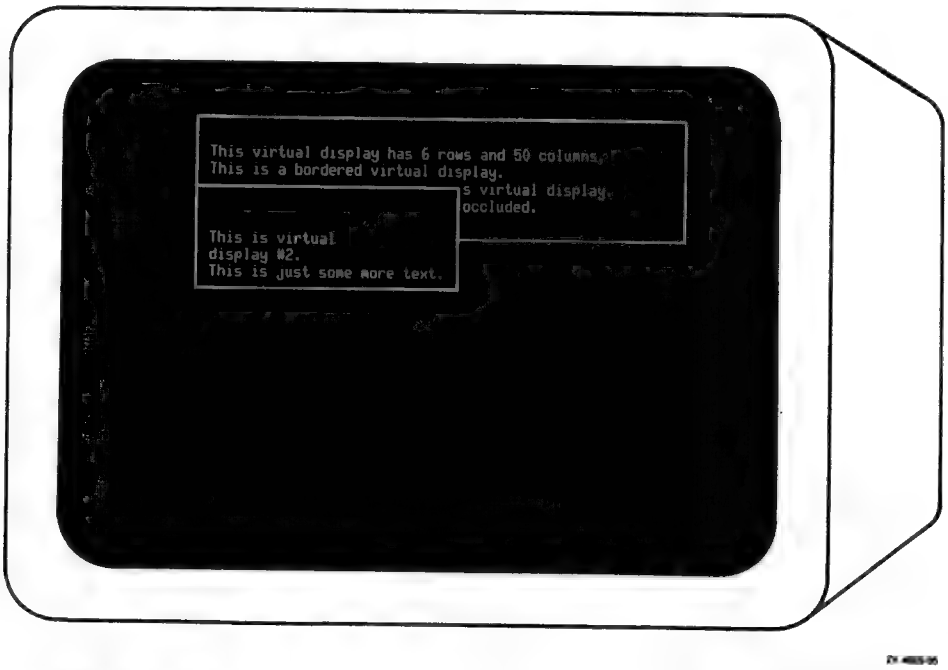
The first virtual display created by this FORTRAN example is shown in Figure RTL–9.1.

Figure RTL–9.1 First Virtual Display Generated by This Example



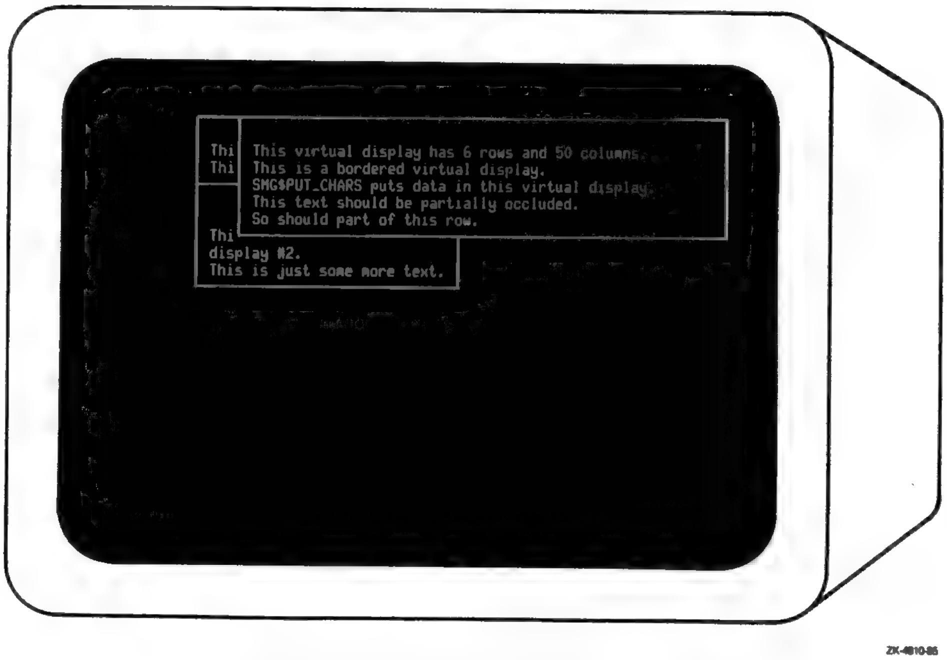
The second virtual display created by this FORTRAN example is shown in Figure RTL–9.2.

Figure RTL–9.2 Second Virtual Display Generated by This Example



The output generated after the call to SMG$COPY_VIRTUAL_DISPLAY is shown in Figure RTL–9.3.

Figure RTL–9.3 Output Generated After the Call to SMG$COPY_VIRTUAL_DISPLAY



SMG$CREATE_KEY_TABLE—Create Key Table

SMG$CREATE_KEY_TABLE creates a table for key definitions.

FORMAT

SMG$CREATE_KEY_TABLE *new-key-table-id*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENT

new-key-table-id

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by reference**

Receives the identifier of the newly-created key table. The **new-key-table-id** argument is the address of an unsigned longword into which is written the key table identifier.

DESCRIPTION

SMG$CREATE_KEY_TABLE creates a key definition table. Key definitions can then be added to this table with the SMGADD_KEY_DEF, SMGLOAD_KEY_DEFS and SMG$DEFINE_KEY routines.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_WRONUMARG	Wrong number of arguments.
LIB$_INSVIRMEM	Insufficient virtual memory.

SMG$CREATE_PASTEBOARD Create Pasteboard

SMG$CREATE_PASTEBOARD creates a pasteboard and returns its assigned pasteboard-id.

FORMAT

SMG$CREATE_PASTEBOARD *new-pasteboard-id [,output-device] [,pb-rows] [,pb-columns] [,preserve-screen-flag]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

new-pasteboard-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Receives the identifier of the newly created pasteboard. The **new-pasteboard-id** argument is the address of an unsigned longword into which is written the new pasteboard identifier.

output-device

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Specifies the file specification or logical name to which the output associated with this pasteboard will be written. The **output-device** argument is the address of a descriptor that points to the name of the output device. If omitted, output is sent to SYS$OUTPUT.

pb-rows

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **write only**
mechanism: **by reference**

Receives the number of rows on the device specified in the **output-device** argument. The **pb-rows** argument is the address of a signed longword integer into which is written the number of rows on the specified device.

pb-columns

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **write only**
mechanism: **by reference**

Receives the number of columns on the device specified in the **output-device** argument. The **pb-columns** argument is the address of a signed longword integer into which is written the number of columns on the specified device.

preserve-screen-flag

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies whether the screen is cleared before the Screen Management Facility performs any output to it. The **preserve-screen-flag** argument is the address of an unsigned longword that contains the screen preservation flag. If **preserve-screen-flag** is set to 0, the screen is initially cleared; if set to 1, the screen is not initially cleared. The default action is to clear the screen. The Screen Management Facility works best when it can manage the entire screen. Therefore, setting this flag to 1 is discouraged.

DESCRIPTION

SMG$CREATE_PASTEBOARD creates a new pasteboard and returns its assigned pasteboard-id. Note that if you request a pasteboard on a device which already has a pasteboard assigned, this routine returns the pasteboard-id of the existing pasteboard and returns the SMG$_PASALREXI status code. Modularity considerations dictate that if the pasteboard already exists, you must not delete it.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_PASALREXI	Successful completion. A pasteboard already exists for this device.
LIB$_INSVIRMEM	Insufficient virtual memory.
SMG$_WRONUMARG	Wrong number of arguments.

EXAMPLES

1

```
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
     C           CREPAS    EXTRN'SMG$CREATE_PASTEBOARD'
     C           CREDIS    EXTRN'SMG$CREATE_VIRTUAL_DISPLAY'
     C           PUTCHA    EXTRN'SMG$PUT_CHARS'
     C           PASDIS    EXTRN'SMG$PASTE_VIRTUAL_DISPLAY'
     C                     Z-ADD0         ZERO    90
     C                     Z-ADD1         LINCOL  90
     C                     Z-ADD2         LINE    90
     C                     Z-ADD5         COLUMN  90
     C                     MOVE 'Menu'    OUT     4
     C* Create the pasteboard.
     C                     CALL CREPAS
     C                     PARM           PASTID  90 WL
     C                     PARMV          ZERO
     C                     PARM           HEIGHT  90 WL
     C                     PARM           WIDTH   90 WL
```

```
C* Create the virtual display.
C                    CALL CREDIS
C                    PARM         HEIGHT     RL
C                    PARM         WIDTH      RL
C                    PARM         DISPID  90 WL
C* Output the 'Menu'.
C                    CALL PUTCHA
C                    PARM         DISPID     RL
C                    PARMD        OUT
C                    PARM         LINE       RL
C                    PARM         COLUMN     RL
C* Paste the virtual display.
C                    CALL PASDIS
C                    PARM         DISPID     RL
C                    PARM         PASTID     RL
C                    PARM         LINCOL     RL
C                    PARM         LINCOL     RL
C                    SETON                   LR
```

The RPG II program above displays 'Menu' beginning at line 2, column 5.

2

```
C+                                                  SMG1.FOR
C This FORTRAN exmaple program demonstrates how to use
C SMG$CREATE_PASTEBOARD.
C-
        IMPLICIT INTEGER*4 (A-Z)
        SMG$M_BOLD = 1
        SMG$M_REVERSE = 2
        SMG$M_BLINK = 4
        SMG$M_UNDERLINE = 8
C+
C Establish the terminal screen as a pasteboard
C by calling SMG$CREATE_PASTEBOARD.
C-
        STATUS = SMG$CREATE_PASTEBOARD (NEW_PID,,,)
        IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Establish a virtual display region by calling
C SMG$CREATE_VIRTUAL_DISPLAY.
C-
        STATUS = SMG$CREATE_VIRTUAL_DISPLAY (5,80,DISPLAY_ID,,,)
        IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Paste the virtual display to the screen, starting at
C row 10, column 15 using SMG$PASTE_VIRTUAL_DISPLAY.
C-
        STATUS = SMG$PASTE_VIRTUAL_DISPLAY(DISPLAY_ID,NEW_PID,10,15)
        IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Write three lines to the screen using SMG$PUT_LINE.
C-
        STATUS = SMG$PUT_LINE (DISPLAY_ID,'This line is underlined',2,
        1                      SMG$M_UNDERLINE,0,,)
        IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
        STATUS = SMG$PUT_LINE (DISPLAY_ID,'This line is blinking',2,
        1                      SMG$M_BLINK,0,,)
        IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
        STATUS = SMG$PUT_LINE (DISPLAY_ID,'This line is reverse video',2,
        1                      SMG$M_REVERSE,0,,)
        IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
        END
```

This FORTRAN program calls Run-Time Library Screen Management routines to format screen output.

SMG$CREATE_VIRTUAL_DISPLAY Create Virtual Display

SMG$CREATE_VIRTUAL_DISPLAY creates a virtual display and returns its assigned display id.

FORMAT

SMG$CREATE_VIRTUAL_DISPLAY *num-rows, num-columns, new-display-id [,display-attributes] [,video-attributes] [,char-set]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

num-rows

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the number of rows in the newly created virtual display. The **num-rows** argument is the address of a signed longword integer that contains the desired number of rows.

num-columns

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the number of columns in the newly created virtual display. The **num-columns** argument is the address of a signed longword integer that contains the desired number of columns.

new-display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Receives the display-id of the newly created virtual display. The **display-id** argument is the address of an unsigned longword into which is written the display identifier.

display-attributes

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies whether the virtual display is bordered (the default is no border). The **display-attributes** argument is the address of an unsigned longword that contains the display attributes mask.

To specify a bordered display, use the mask SMG$M_BORDER. If omitted, the display is not bordered. Two other display attributes may also be specfied. If SMG$M_TRUNC_ICON is specified, an icon (generally a diamond shape) is displayed where truncation has occured by exceeding the dimensions of the virtual display. When SMG$M_DISPLAY_CONTROLS is used, control characters such as carriage return and line feed are displayed as characters (<CR> <LF>) so that you can easily see where they are.

video-attributes

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the default rendition to be applied to all output in this virtual display unless overridden by a call to a specific output routine (for example, SMG$CHANGE_RENDITION). The **video-attributes** argument is the address of an unsigned longword that contains the video attributes mask.

Allowed values for this argument are as follows:

SMG$M_BLINK	Displays blinking characters
SMG$M_BOLD	Displays characters in higher-than-normal intensity
SMG$M_REVERSE	Displays characters in reverse video, that is, using the opposite default rendition of the virtual display
SMG$M_UNDERLINE	Displays underlined characters

char-set

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the default character set for all text in this virtual display. The **char-set** argument is the address of an unsigned longword that contains the character set specifier. At this time, the only valid value is SMG$C_ASCII, which is also the default.

DESCRIPTION

SMG$CREATE_VIRTUAL_DISPLAY creates a new virtual display and returns its display id. Initially, the virtual display contains blanks, and the virtual cursor is positioned at row 1, column 1.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
LIB$_INSVIRMEM	Insufficient virtual memory.
SMG$_INVARG	Invalid argument. **Video-attributes** or **display-attributes** contains an unknown value.
SMG$_WRONUMARG	Wrong number of arguments.

EXAMPLES

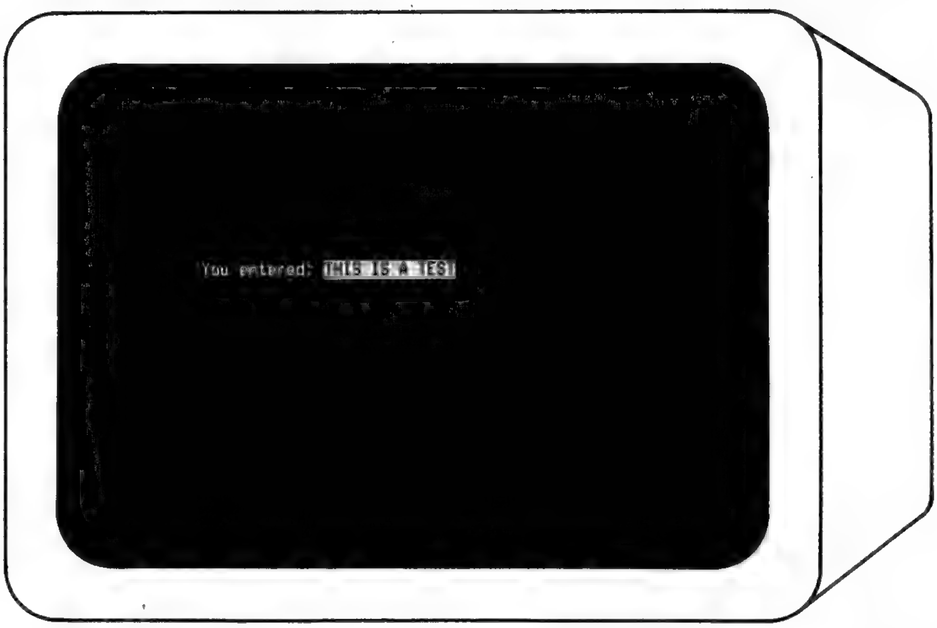
1

```
C+
C This FORTRAN example program demonstrates how to use
C SMG$CREATE_VIRTUAL_DISPLAY.
C-
        IMPLICIT INTEGER*4 (A-Z)
        CHARACTER*80    OUT_STR,TRIM_STR
        CHARACTER*18    PROMPT          /'Please enter data '/
        SMG$M_BOLD = 1
        SMG$M_REVERSE = 2
        SMG$M_BLINK = 4
        SMG$M_UNDERLINE = 8
C+
C Establish the terminal keyboard as the virtual keyboard
C by calling SMG$CREATE_VIRTUAL_KEYBOARD.
C-
        STATUS = SMG$CREATE_VIRTUAL_KEYBOARD(KEYBOARD_ID,,,)
        IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Establish the terminal screen as a pasteboard using
C SMG$CREATE_PASTEBOARD.
C-
        STATUS = SMG$CREATE_PASTEBOARD (NEW_PID,,,)
        IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Establish a virtual display region by
C calling SMG$CREATE_VIRTUAL_DISPLAY.
C-
        STATUS = SMG$CREATE_VIRTUAL_DISPLAY (5,80,DISPLAY_ID,,,)
        IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Paste the virtual display to the screen, starting at
C row 10, column 15.  To paste the virtual display, use
C SMG$PASTE_VIRTUAL_DISPLAY.
C-
        STATUS = SMG$PASTE_VIRTUAL_DISPLAY(DISPLAY_ID,NEW_PID,10,15)
        IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Prompt the user for input, and accept that input using
C SMG$READ_STRING.
C-
        STATUS = SMG$READ_STRING(KEYBOARD_ID,OUT_STR,PROMPT,,,,,,,)
        IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Clear the screen using SMG$ERASE_PASTEBOARD.
C-
        STATUS = SMG$ERASE_PASTEBOARD (NEW_PID)
        IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Trim any trailing blanks from the user input
C by calling STR$TRIM.
C-
        STATUS = STR$TRIM(TRIM_STR,OUT_STR,STR_LEN)
        IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
```

```
C Display the data input by the user using SMG$PUT_CHARS
C and SMG$PUT_LINE.
C-
        STATUS = SMG$PUT_CHARS(DISPLAY_ID,'You entered: ',,,,,,)
        IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
        STATUS = SMG$PUT_LINE(DISPLAY_ID,TRIM_STR(1:STR_LEN),,
        1                    SMG$M_REVERSE,0,,)
        IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
        END
```

The output generated by this FORTRAN example is shown in Figure RTL-10.

Figure RTL-10 Output of FORTRAN Program Calling SMG$CREATE_VIRTUAL_DISPLAY



2

For an example of calling SMG$CREATE_VIRTUAL_DISPLAY in RPG, see the example in the description of SMG$CREATE_PASTEBOARD.

SMG$CREATE_VIRTUAL_KEYBOARD Create Virtual Keyboard

SMG$CREATE_VIRTUAL_KEYBOARD creates a virtual keyboard and returns its assigned keyboard-id.

FORMAT

SMG$CREATE_VIRTUAL_KEYBOARD *new-keyboard-id [,filespec] [,default-filespec] [,resultant-filespec] [,recall-size]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

new-keyboard-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Receives the keyboard identifier of the newly created virtual keyboard. The **new-keyboard-id** argument is the address of an unsigned longword into which is written the keyboard identifier.

filespec

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

String containing the file specification or logical name of the file or terminal to be used for this virtual keyboard. The **filespec** argument is the address of a descriptor pointing to the file specification. If omitted, this defaults to SYS$INPUT.

default-filespec

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

String containing the default file specification. The **default-filespec** argument is the address of a descriptor pointing to the default file specification. If omitted, the null string is used.

Default-filespec might be used to specify a default device and directory, leaving the **filespec** argument to supply the file name and type.

resultant-filespec

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

String into which the procedure writes the fully expanded file specification of the file used. The **resultant-filespec** argument is the address of a descriptor pointing to the string into which is written the file specification that was used.

recall-size

VMS Usage: **byte_unsigned**
type: **byte (unsigned)**
access: **read only**
mechanism: **by reference**

Number of input lines to be saved for later recall. The optional **recall-size** argument is the address of an unsigned byte containing the specified number of lines. A value of zero turns off input line recall. By default, 20 lines are saved for later recall.

DESCRIPTION

SMG$CREATE_VIRTUAL_KEYBOARD creates the association between a file specification (terminal name or RMS file) and a virtual keyboard. The keyboard identifier is then passed to other SMG$ procedures in order to identify the input stream being acted upon.

If **filespec** does not refer to a terminal, the file is opened using RMS and all further access to that file is performed through RMS. If **filespec** is a terminal, this procedure assigns a channel to the terminal and sets the terminal's keyboard to application mode (if supported). These attributes are restored to their previous values when the virtual keyboard is deleted. The virtual keyboard is deleted automatically when the image exits and can also be deleted by a call to SMG$DELETE_VIRTUAL_KEYBOARD.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_FILTOOLON	File specification is too long (over 255 characters).
SMG$_WRONUMARG	Wrong number of arguments.
LIB$_INSEF	Insufficient number of event flags.
LIB$_INSVIRMEM	Insufficient virtual memory.
LIB$_INVSTRDES	Invalid string descriptor.

Any RMS condition values returned by $OPEN or $CONNECT.

Any condition values returned by $GETDVIW, $ASSIGN, or $DCLEXH.

EXAMPLE

```
C+
C This FORTRAN example program demonstrates the use of
C SMG$CREATE_VIRTUAL_KEYBOARD, SMG$CREATE_KEY_TABLE,
C SMG$ADD_KEY_DEF, and SMG$READ_COMPOSED_LINE.
C-

        INTEGER SMG$CREATE_VIRTUAL_KEYBOARD, SMG$CREATE_KEY_TABLE
        INTEGER SMG$ADD_KEY_DEF, SMG$READ_COMPOSED_LINE
        INTEGER SMG$DELETE_KEY_DEF, KEYBOARD, KEYTABLE, STATUS

C+
C Include the SMG definitions. In particular, we want SMG$M_KEY_NOECHO
C and SMG$M_KEY_TERMINATE.
C-
        INCLUDE '($SMGDEF)'

C+
C Create a virtual keyboard (using SMG$CREATE_VIRTUAL_KEYBOARD)
C and create a key table (using SMG$CREATE_KEY_TABLE).
C-
        STATUS = SMG$CREATE_VIRTUAL_KEYBOARD (KEYBOARD)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

        STATUS = SMG$CREATE_KEY_TABLE (KEYTABLE)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Prompt the user with the following instructions.
C-
        WRITE (6,*) 'When you see the prompt (->), strike the following'
        WRITE (6,*) 'keys (on the KEYPAD): '
        WRITE (6,*) '            PF1 '
        WRITE (6,*) '            5 '
        WRITE (6,*) '            PF3 '
        WRITE (6,*) ' '
        WRITE (6,*) 'When you have done this, the following sentence'
        WRITE (6,*) '(and nothing more) should appear following the'
        WRITE (6,*) 'prompt: '
        WRITE (6,*) '(PF3 should act as a carriage return.)'
        WRITE (6,*) ' '
        WRITE (6,*) 'NOW IS THE TIME FOR ALL TEXT TO APPEAR.'

C+
C Add key definitions by calling SMG$ADD_KEY_DEF.
C-
        STATUS = SMG$ADD_KEY_DEF (KEYTABLE, 'PF1', , ,
     1  'NOW IS THE TIME FOR ')
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

        STATUS = SMG$ADD_KEY_DEF (KEYTABLE, 'KP5', , ,
     1  'TEXT TO APPEAR.')
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

        STATUS = SMG$ADD_KEY_DEF (KEYTABLE, 'PF3', ,
     1  SMG$M_KEY_NOECHO + SMG$M_KEY_TERMINATE ,
     1  'THIS SHOULD NOT BE ECHOED.  IF YOU CAN
     1  SEE THIS, AN ERROR EXISTS.')
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Call SMG$READ_COMPOSED_LINE to read a line of input.
C-
        WRITE(6,*) ' '
        STATUS = SMG$READ_COMPOSED_LINE (KEYBOARD, KEYTABLE, R_TEXT,
     1          '->')
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

        END
```

Output session:

```
$ RUN example
```

When you see the prompt (->), strike the following keys (on the KEYPAD):

- PF1
- 5
- PF3

After entering PF3, the following sentence should appear following the prompt:

```
NOW IS THE TIME FOR ALL TEXT TO APPEAR.
->NOW IS THE TIME FOR ALL TEXT TO APPEAR.
$
```

SMG$CURSOR_COLUMN Return Cursor Column Position

SMG$CURSOR_COLUMN returns the virtual cursor's current column position in a specified virtual display.

FORMAT

SMG$CURSOR_COLUMN *display-id*

RETURNS

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

SMG$CURSOR_COLUMN returns the current virtual cursor column position.

ARGUMENT

display-id
VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**
The display for which the column position is returned. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

Display-id is returned by SMG$CREATE_VIRTUAL_DISPLAY.

DESCRIPTION

SMG$CURSOR_COLUMN returns a longword containing the value of the current virtual cursor column position for the specified virtual display. If the **display-id** is omitted, this routine signals SMG$_WRONUMARG. If the **display-id** is invalid, this routine signals SMG$_INVDIS_ID.

CONDITION VALUES SIGNALED

SMG$_INVDIS_ID	Invalid **display-id**
SMG$_WRONUMARG	Wrong number of arguments

SMG$CURSOR_ROW—Return Cursor Row Position

SMG$CURSOR_ROW returns the virtual cursor's current row position in a specified virtual display.

FORMAT

SMG$CURSOR_ROW *display-id*

RETURNS

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

SMG$CURSOR_ROW returns the current row position.

ARGUMENT

display-id
VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

The display for which the row position is returned. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

Display-id is returned by SMG$CREATE_VIRTUAL_DISPLAY.

DESCRIPTION

SMG$CURSOR_ROW returns a longword containing the value of the current virtual cursor row position for the specified virtual display.

CONDITION VALUES SIGNALED

SMG$_INVDIS_ID	Invalid **display-id**
SMG$_WRONUMARG	Wrong number of arguments

SMG$DEFINE_KEY—Perform a DEFINE /KEY Command

SMG$DEFINE_KEY performs the DEFINE/KEY command you provide.

FORMAT

SMG$DEFINE_KEY *key-table-id ,command-line*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

key-table-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Identification of the key definition table for which the DEFINE/KEY command is to be performed. The **key-table-id** argument is the address of an unsigned longword that contains the key table identifier.

Key-table-id is returned by SMG$CREATE_KEY_TABLE.

command-line

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

String containing the DEFINE/KEY command to be performed. The **command-line** argument is the address of a descriptor pointing to the command to be performed.

The valid qualifiers for the DEFINE/KEY command are:

- /PROTECT
- /TERMINATE
- /NOECHO
- /LOCK

DESCRIPTION

SMG$DEFINE_KEY parses and performs a DEFINE/KEY command. It can be used by programs that accept DEFINE/KEY commands but do not parse the commands themselves.

SMG$DEFINE_KEY calls CLI$DCL_PARSE to parse the command line and then makes the appropriate call to SMG$ADD_KEY_DEF. Use of this procedure requires that the image be run under the DCL Command Language Interpreter.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_WRONUMARG	Wrong number of arguments.

Any condition values returned by LIB$SCOPY_DXDX.

Any condition values returned by CLI$ routines.

Any condition values returned by SMG$ADD_KEY_DEF.

SMG$DEL_TERM_TABLE—Delete Terminal Table

SMG$DEL_TERM_TABLE terminates access to TERMTABLE.EXE and frees the associated virtual address space.

FORMAT

SMG$DEL_TERM_TABLE

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

None.

DESCRIPTION

SMG$DEL_TERM_TABLE terminates access to TERMTABLE.EXE. Calling this routine is optional. This routine is useful only in the case where a calling program might need to reuse the virtual address space used by a private TERMTABLE.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.

SMG$DELETE_CHARS—Delete Characters

SMG$DELETE_CHARS deletes characters in a virtual display.

FORMAT

SMG$DELETE_CHARS *display-id ,num-chars ,row ,column*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Identifies the virtual display from which characters are to be deleted. The **display-id** argument is the address of an unsigned longword integer that contains the display identifier.

Display-id is returned by SMG$CREATE_VIRTUAL_DISPLAY.

num-chars

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the number of characters to be deleted. The **num-chars** argument is the address of a signed longword integer that contains the number of characters to be deleted.

row

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the row position at which to start the deletion. The **row** argument is the address of a signed longword integer that contains the row number at which to start the deletion.

column

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the column position at which to start the deletion. The **column** argument is the address of a signed longword integer that contains the column position at which to start the deletion.

DESCRIPTION

SMG$DELETE_CHARS deletes a specified number of characters, starting at a specified row and column position. Remaining characters on the line are shifted to the left to occupy the vacated space(s). Note that this routine deletes characters only on a single line.

If you specify more characters than are available for deletion, SMG$DELETE_CHARS deletes all characters from the specified column position to the end of the line.

This routine leaves the virtual cursor at the position of the first character deleted.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVROW	Invalid row position. The specified row is outside the virtual display.
SMG$_INVCOL	Invalid column position. The specified column is outside the virtual display.
SMG$_INVDIS_ID	Invalid **display-id**.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_INVARG	Invalid argument. The number of characters specified extends outside virtual display.

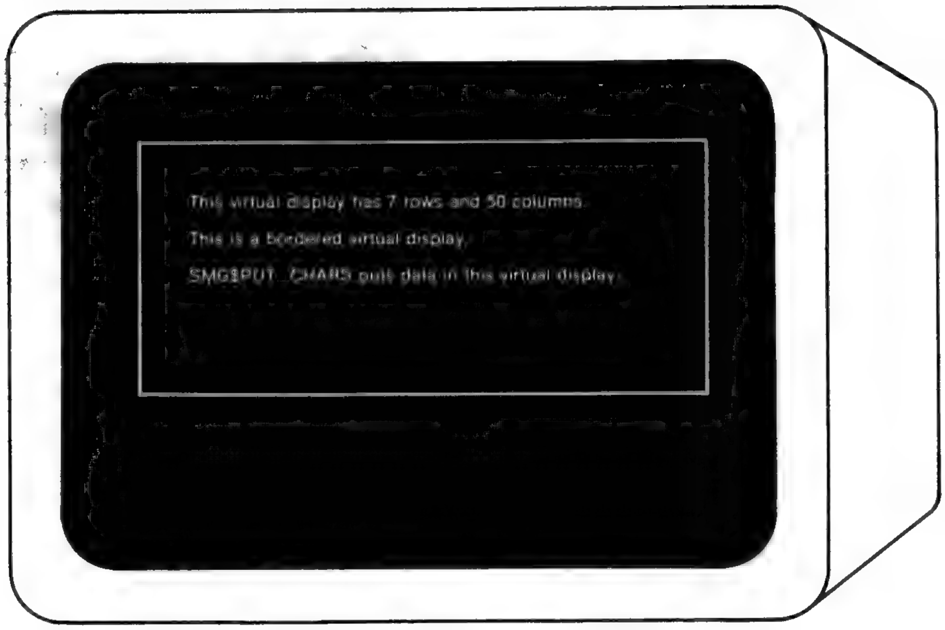
EXAMPLE

```
C+
C This FORTRAN example program demonstrates the use of
C SMG$DELETE_CHARS.
C-
        INTEGER SMG$CREATE_VIRTUAL_DISPLAY, SMG$CREATE_PASTEBOARD
        INTEGER SMG$PASTE_VIRTUAL_DISPLAY,  SMG$PUT_CHARS, SMG$DELETE_CHARS
        INTEGER DISPLAY1, PASTE1, ROWS, COLUMNS, BORDER
C+
C Create the virtual display be calling SMG$CREATE_VIRTUAL_DISPLAY.
C To give it a border, set BORDER = 1. No border would be BORDER = 0.
C-
        ROWS = 7
        COLUMNS = 50
        BORDER = 1
        ISTATUS = SMG$CREATE_VIRTUAL_DISPLAY
     1                          (ROWS, COLUMNS, DISPLAY1, BORDER)
        IF (.NOT. ISTATUS) WRITE (6, 900) 'SMG$CREATE_VIRTUAL_DISPLAY', ISTATUS
C+
C Call SMG$CREATE_PASTEBOARD to create the pasteboard.
C-
        ISTATUS = SMG$CREATE_PASTEBOARD (PASTE1)
        IF (.NOT. ISTATUS) WRITE (6, 900) 'SMG$CREATE_PASTEBOARD', ISTATUS
C+
C Use SMG$PUT_CHARS to put data in the virtual display.
C-
        ISTATUS = SMG$PUT_CHARS ( DISPLAY1,
     1          ' This virtual display has 7 rows and 50 columns.', 2, 1)
        ISTATUS = SMG$PUT_CHARS ( DISPLAY1,
     1          ' This is a bordered virtual display.', 4, 1)
        ISTATUS = SMG$PUT_CHARS ( DISPLAY1,
     1          ' SMG$PUT_CHARS puts data in this virtual display.', 6, 1)
C+
C Paste the virtual display to the pasteboard using
C SMG$PASTE_VIRTUAL_DISPLAY.
C-
```

```
      ISTATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DISPLAY1, PASTE1, 4, 15)
900   FORMAT (' Routine ', A, ' returned a status of ', Z8)
C+
C Call SMG$DELETE_CHARS to delete 4 characters from row 4
C starting from character (column) 14, removing the characters
C "rder" from the word "bordered".
C-
      ISTATUS = SMG$DELETE_CHARS ( DISPLAY1, 4, 4, 14)
      END
```

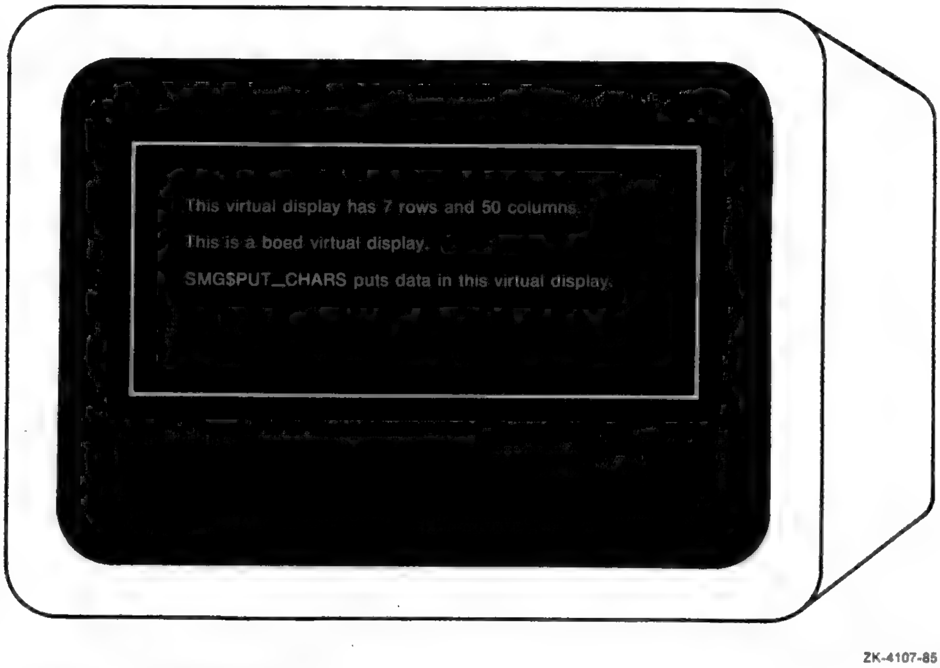
The output generated by this FORTRAN program before the call to SMG$DELETE_CHARS is shown in Figure RTL-11.

Figure RTL-11 Output Generated Before the Call to SMG$DELETE_CHARS



The output generated after the call to SMG$DELETE_CHARS is shown in Figure RTL-12.

Figure RTL-12 Output Generated After the Call to SMG$DELETE_CHARS



SMG$DELETE_KEY_DEF—Delete Key Definition

SMG$DELETE_KEY_DEF deletes a key definition from a specified table of key definitions.

FORMAT

SMG$DELETE_KEY_DEF *key-table-id ,key-name [,if-state]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

key-table-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Identifies the key table from which the key definition is deleted. The **key-table-id** argument is the address of an unsigned longword that contains the key table identifier.

key-name

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

String containing the name of the key whose definition is to be deleted. The **key-name** argument is the address of a descriptor pointing to the key name. **Key-name** is stripped of trailing blanks and converted to uppercase before use.

Table 3–1 in Part I of this manual lists the valid key names.

if-state

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

String containing a state name which further qualifies key-name. The **if-state** argument is the address of a descriptor pointing to the state name. If omitted, the null state is used. Thus if a key has several definitions depending on various values of **if-state**, this routine lets you delete only one of those definitions.

DESCRIPTION

SMG$DELETE_KEY_DEF deletes a key definition from a specified table of key definitions.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVKTB_ID	Invalid **key-table-id**.
SMG$_KEYNOTDEF	Key is not defined.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_KEYDEFPRO	Key definition is protected.

SMG$DELETE_LINE—Delete Line

SMG$DELETE_LINE deletes lines from a virtual display.

FORMAT

SMG$DELETE_LINE *display-id ,start-line [,number-lines]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Identifies the virtual display from which lines are to be deleted. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

Display-id is returned by SMG$CREATE_VIRTUAL_DISPLAY.

start-line

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the first line to be deleted from the virtual display. The **start-line** argument is the address of a signed longword integer that contains the number of the first line to be deleted.

number-lines

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the number of lines to be deleted. The **number-lines** argument is the address of a signed longword integer that contains the number of lines to be deleted. If omitted, one line is deleted.

DESCRIPTION

SMG$DELETE_LINE deletes one or more lines from a virtual display and scrolls the remaining lines up into the space created by the deletion. Blank lines fill the display on the bottom. The virtual cursor is left at the first column position in **start-line**.

CONDITION VALUES RETURNED

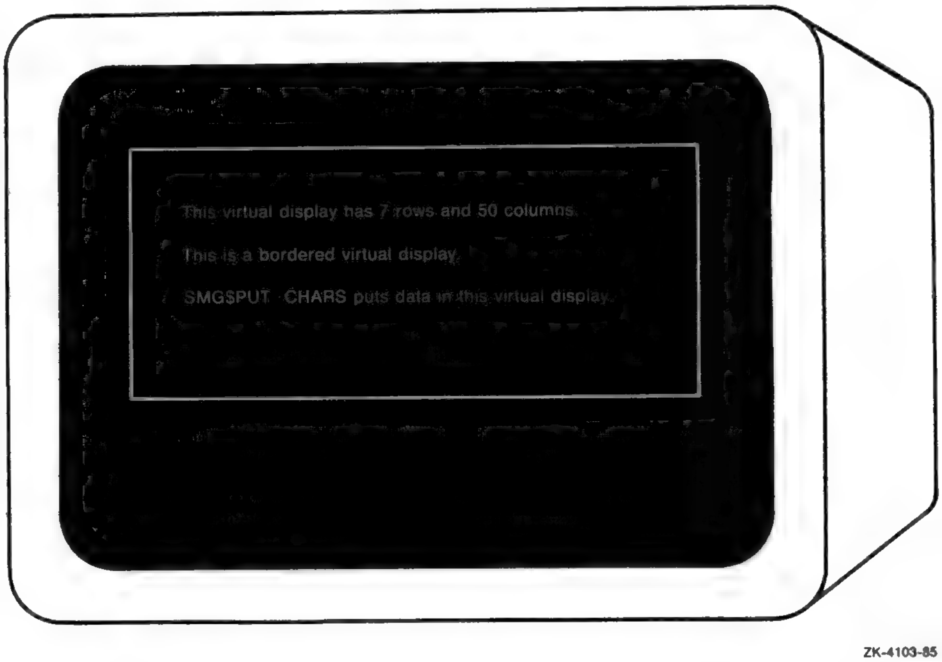
SS$_NORMAL	Normal successful completion.
SMG$_INVDIS_ID	Invalid **display-id**.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_INVROW	Invalid row.
SMG$_INVARG	Invalid argument.

EXAMPLE

```
C+
C This FORTRAN example program demonstrates the use of SMG$DELETE_LINE.
C-
        INTEGER SMG$CREATE_VIRTUAL_DISPLAY, SMG$CREATE_PASTEBOARD
        INTEGER SMG$PASTE_VIRTUAL_DISPLAY,  SMG$PUT_CHARS, SMGDELETE_LINE
        INTEGER DISPLAY1, PASTE1, ROWS, COLUMNS, BORDER
C+
C Create the virtual display by calling SMG$CREATE_VIRTUAL_DISPLAY.
C To give it a border, set BORDER = 1.  No border would be BORDER = 0.
C-
        ROWS = 7
        COLUMNS = 50
        BORDER = 1
        ISTATUS = SMG$CREATE_VIRTUAL_DISPLAY
     1                          (ROWS, COLUMNS, DISPLAY1, BORDER)
        IF (.NOT. ISTATUS) WRITE (6, 900) 'SMG$CREATE_VIRTUAL_DISPLAY', ISTATUS
C+
C Call SMG$CREATE_PASTEBOARD to create the pasteboard.
C-
        ISTATUS = SMG$CREATE_PASTEBOARD (PASTE1)
        IF (.NOT. ISTATUS) WRITE (6, 900) 'SMG$CREATE_PASTEBOARD', ISTATUS
C+
C Use SMG$PUT_CHARS to put data in the virtual display.
C-
        ISTATUS = SMG$PUT_CHARS ( DISPLAY1,
     1          ' This virtual display has 7 rows and 50 columns.', 2, 1)
        ISTATUS = SMG$PUT_CHARS ( DISPLAY1,
     1          ' This is a bordered virtual display.', 4, 1)
        ISTATUS = SMG$PUT_CHARS ( DISPLAY1,
     1          ' SMG$PUT_CHARS puts data in this virtual display.', 6, 1)
C+
C Paste the virtual display to the pasteboard using
C SMG$PASTE_VIRTUAL_DISPLAY.
C-
        ISTATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DISPLAY1, PASTE1, 4, 15)
900     FORMAT (' Routine ', A, ' returned a status of ', Z8)
C+
C Call SMG$DELETE_LINE to delete rows 3, 4, and 5.
C-
        ISTATUS = SMG$DELETE_LINE ( DISPLAY1, 3, 3)
        END
```

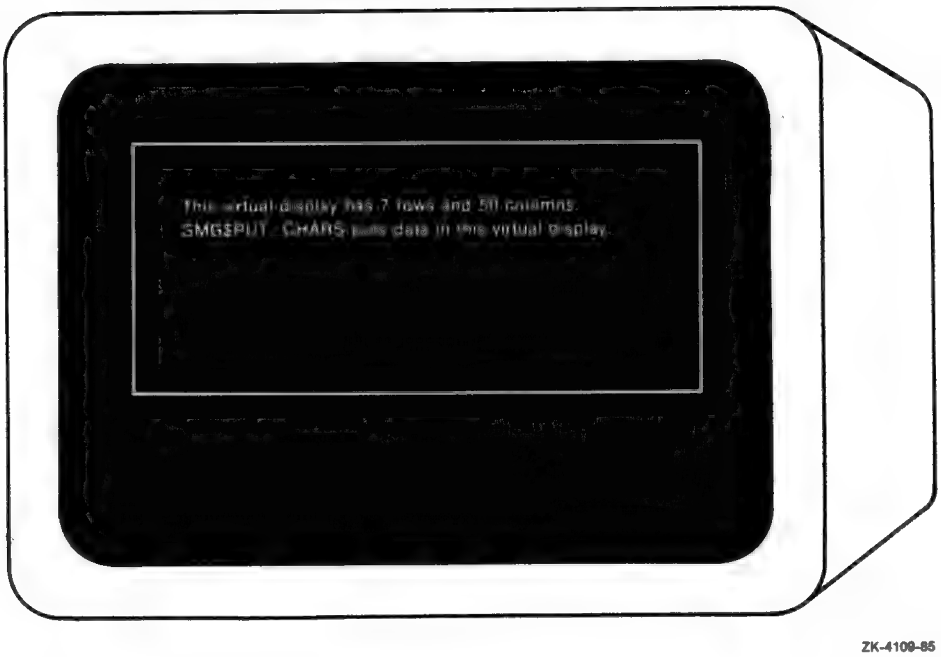
The output generated by this FORTRAN program before the call to SMG$DELETE_LINE is shown in Figure RTL–13.

Figure RTL–13 Output Generated by FORTRAN Program Before the Call to SMG$DELETE_LINE



The output generated after the call to SMG$DELETE_LINE is shown in Figure RTL–14.

Figure RTL–14 Output Generated After the Call to SMG$DELETE_LINE



SMG$DELETE_PASTEBOARD—Delete Pasteboard

SMG$DELETE_PASTEBOARD deletes a pasteboard.

FORMAT

SMG$DELETE_PASTEBOARD *pasteboard-id [,clear-screen-flag]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

pasteboard-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the pasteboard to be deleted. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

Pasteboard-id is returned by SMG$CREATE_PASTEBOARD.

clear-screen-flag

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Determines whether this routine clears the screen after deleting the specified pasteboard. The **clear-screen-flag** argument is the address of an unsigned longword that contains the flag. If **clear-screen-flag** is 1, the screen is cleared; if 0, the screen is not cleared. If this argument is omitted, the default is to clear the screen.

DESCRIPTION

SMG$DELETE_PASTEBOARD flushes all output to the display, terminates all use of the specified pasteboard, and deallocates all resources associated with the pasteboard.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_INVPAS_ID	Invalid **pasteboard-id**.
SMG$_NOTPASTED	The specified virtual display is not pasted to the specified pasteboard.

Any condition values returned by $DASSGN, LIB$FREE_VM, LIB$FREE_EF, or SMG$FLUSH_BUFFER.

SMG$DELETE_VIRTUAL_DISPLAY
Delete Virtual Display

SMG$DELETE_VIRTUAL_DISPLAY deletes a virtual display.

FORMAT

SMG$DELETE_VIRTUAL_DISPLAY *display-id*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENT

display-id
VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**
Specifies the virtual display to be deleted. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

Display-id is returned by SMG$CREATE_VIRTUAL_DISPLAY.

DESCRIPTION

SMG$DELETE_VIRTUAL_DISPLAY deletes a virtual display and removes it from any pasteboard on which it is pasted. It also deallocates any buffer space associated with the virtual display.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVDIS_ID	Invalid **display-id**.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_NOTPASTED	The specified virtual display is not pasted to the specified pasteboard.

Any condition values returned by LIB$FREE_VM.

SMG$DELETE_VIRTUAL_KEYBOARD Delete Virtual Keyboard

SMG$DELETE_VIRTUAL_KEYBOARD deletes a virtual keyboard.

FORMAT

SMG$DELETE_VIRTUAL_KEYBOARD *keyboard-id*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENT

keyboard-id

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Specifies the virtual keyboard to be deleted. The **keyboard-id** argument is the address of an unsigned longword that contains the keyboard identifier.

Keyboard-id is returned by SMG$CREATE_VIRTUAL_KEYBOARD.

DESCRIPTION

SMG$DELETE_VIRTUAL_KEYBOARD deletes a virtual keyboard. Any terminal attributes specified when the keyboard was created are reset to their previous values and the keypad mode (numeric or application) is reset to its original state. In addition, the channel is deassigned and, if the virtual keyboard was a file, the file is closed.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_NOT_A_TRM	Informational message. The device was not a terminal.
SMG$_INVKBD_ID	Invalid **keyboard-id**.
SMG$_WRONUMARG	Wrong number of arguments.

SMG$DISABLE_BROADCAST_TRAPPING Disable Broadcast Trapping

SMG$DISABLE_BROADCAST_TRAPPING disables trapping of broadcast messages for the specified terminal.

FORMAT

SMG$DISABLE_BROADCAST_TRAPPING

pasteboard-id

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

pasteboard-id
VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the pasteboard for the terminal to be affected. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

DESCRIPTION

SMG$DISABLE_BROADCAST_TRAPPING disables trapping of broadcast messages for the specified terminal. SMG$DISABLE_BROADCAST_TRAPPING deassigns the mailbox set with SMG$SET_BROADCAST_TRAPPING, resets the terminal characteristics, and therefore allows the user to SPAWN a subprocess. This routine must be used to disable any broadcast trapping set with the routine SMG$SET_BROADCAST_TRAPPING.

Note that if both broadcast trapping and the trapping of unsolicited input are enabled, then both SMG$DISABLE_BROADCAST_TRAPPING and SMG$DISABLE_UNSOLICITED_INPUT must be invoked to deassign the mailbox.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion
SMG$_WRONUMARG	Wrong number of arguments

Any condition value returned by $QIOW.

EXAMPLE

```
10      !+
        !This program creates three virtual displays on one pasteboard.
        !The first virtual display contains instructions for the user,
        !the second shows trapped unsolicited input, and the third
        !lists trapped broadcast messages. The program sits in an
        !infinite loop until the user types a CTRL/Z.
        !
        !When the program traps unsolicited input, both broadcast message
        !and unsolicited input trapping are disabled, and a subprocess
        !is spawned which executes the trapped user input.
        !
        !When control returns to the main process, broadcast trapping and
        !the trapping of unsolicited input are both reenabled. If the
        !unsolicited input which is trapped is a CTRL/Z, the program exits.
        !-
        OPTION TYPE = EXPLICIT
        !+
        !Declaration of all routines called by the main program.
        !-
        EXTERNAL SUB LIB$STOP (LONG BY VALUE)
        EXTERNAL LONG FUNCTION SMG$CREATE_PASTEBOARD (LONG)
        EXTERNAL LONG FUNCTION SMG$CREATE_VIRTUAL_DISPLAY (LONG, LONG, &
                                   LONG, LONG, LONG)
        EXTERNAL LONG FUNCTION SMG$PASTE_VIRTUAL_DISPLAY (LONG, LONG, &
                                   LONG, LONG)
        EXTERNAL LONG FUNCTION SMG$PUT_LINE (LONG, STRING, LONG, LONG, &
                                   LONG, LONG)
        EXTERNAL LONG FUNCTION SMG$SET_BROADCAST_TRAPPING (LONG, LONG)
        EXTERNAL LONG FUNCTION SMG$CREATE_KEY_TABLE (LONG)
        EXTERNAL LONG FUNCTION SMG$CREATE_VIRTUAL_KEYBOARD (LONG)
        EXTERNAL LONG FUNCTION SMG$LABEL_BORDER (LONG, STRING, LONG, &
                                   LONG, LONG, LONG)
        EXTERNAL LONG FUNCTION SMG$ENABLE_UNSOLICITED_INPUT (LONG, LONG, &
                                   LONG)
        EXTERNAL LONG FUNCTION SMG$DISABLE_UNSOLICITED_INPUT (LONG)
        EXTERNAL LONG FUNCTION SMG$DISABLE_BROADCAST_TRAPPING (LONG)
        EXTERNAL LONG FUNCTION SMG$DELETE_PASTEBOARD (LONG)
        !+
        !Declaration of the two AST routines:
        !GET_MSG is called when a broadcast message is trapped
        !GET_INPUT is called when there is unsolicited input -
        !GET_INPUT is the routine which spawns the subprocess
        !-
        EXTERNAL INTEGER        GET_MSG
        EXTERNAL INTEGER        GET_INPUT
        DECLARE LONG pb_id, ret_status, display_id, display2_id, display3_id, &
                key_id, key_tab_id, counter
        !+
        !Create a MAP area for variables which must be shared between the
        !main program and the AST routines.
        !-
        MAP (params) LONG disp_info(2), LONG keyboard_info(4), LONG done_flag
        DECLARE STRING CONSTANT top_label = "User Input"
        DECLARE STRING CONSTANT ins_label = "Instructions"
        DECLARE STRING CONSTANT msg_label = "Messages"
        DECLARE STRING CONSTANT instr_0 = "Type commands to fill INPUT display."
        DECLARE STRING CONSTANT instr_1 = "Type CTRL/T to fill MESSAGES display."
        DECLARE STRING CONSTANT instr_2 = "Type CTRL/Z to exit."
        DECLARE LONG CONSTANT   advance = 1
        DECLARE LONG CONSTANT   wrap = 1
        !+
        !Declare any external constants used in the program. The values
        !for these constants are assigned when you link the program with
        !SMGDEF.
```

```
!-
EXTERNAL LONG CONSTANT SMG$M_BOLD
EXTERNAL LONG CONSTANT SMG$M_REVERSE
EXTERNAL LONG CONSTANT SMG$M_BLINK
EXTERNAL LONG CONSTANT SMG$M_UNDERLINE
EXTERNAL LONG CONSTANT SMG$M_BORDER
EXTERNAL LONG CONSTANT SMG$_EOF
EXTERNAL LONG CONSTANT SS$_NORMAL
EXTERNAL LONG CONSTANT SMG$_NO_MORMSG
!+
!The done_flag variable is clear (0) unless the user input was
!a CTRL/Z - in that case the program exits.
!-
done_flag = 0
!+
!Create the pasteboard and the virtual keyboard
!-
ret_status = SMG$CREATE_PASTEBOARD (pb_id)
IF (ret_status AND 1%) = 0% THEN
   CALL LIB$STOP(ret_status BY VALUE)
END IF
!+
!This is one of the values which must be stored in the MAP area.
!-
disp_info(0) = pb_id
ret_status = SMG$CREATE_VIRTUAL_KEYBOARD (key_id)
IF (ret_status AND 1%) = 0% THEN
   CALL LIB$STOP(ret_status BY VALUE)
END IF
ret_status = SMG$CREATE_KEY_TABLE (key_tab_id)
IF (ret_status AND 1%) = 0% THEN
   CALL LIB$STOP(ret_status BY VALUE)
END IF

!+
!Create the three virtual displays
!-
ret_status = SMG$CREATE_VIRTUAL_DISPLAY(3 BY REF, 75 BY REF, &
        display3_id, SMG$M_BORDER BY REF, SMG$M_REVERSE BY REF)
IF (ret_status AND 1%) = 0% THEN
   CALL LIB$STOP(ret_status BY VALUE)
END IF
ret_status = SMG$CREATE_VIRTUAL_DISPLAY(6 BY REF, 75 BY REF, &
        display_id, SMG$M_BORDER BY REF, SMG$M_REVERSE BY REF)
IF (ret_status AND 1%) = 0% THEN
   CALL LIB$STOP(ret_status BY VALUE)
END IF
ret_status = SMG$CREATE_VIRTUAL_DISPLAY(6 BY REF, 75 BY REF, &
        display2_id, SMG$M_BORDER BY REF, SMG$M_REVERSE BY REF)
IF (ret_status AND 1%) = 0% THEN
   CALL LIB$STOP(ret_status BY VALUE)
END IF
!+
!The disp_info and keyboard_info arrays are required in the MAP.
!-
disp_info(1) = display2_id

keyboard_info(0) = key_id
keyboard_info(1) = key_tab_id
keyboard_info(2) = display_id
keyboard_info(4) = pb_id
!+
!Put Label borders around the three virtual displays.
!-
ret_status = SMG$LABEL_BORDER (display3_id, ins_label,,, &
        SMG$M_BOLD BY REF, SMG$M_REVERSE BY REF)
IF (ret_status AND 1%) = 0% THEN
```

```
   CALL LIB$STOP(ret_status BY VALUE)
END IF
ret_status = SMG$LABEL_BORDER (display_id, top_label,,, &
        SMG$M_BOLD BY REF,)
IF (ret_status AND 1%) = 0% THEN
   CALL LIB$STOP(ret_status BY VALUE)
END IF
ret_status = SMG$LABEL_BORDER (display2_id, msg_label,,, &
        SMG$M_BOLD BY REF,)
IF (ret_status AND 1%) = 0% THEN
   CALL LIB$STOP(ret_status BY VALUE)
END IF

!+
!Fill the INSTRUCTIONS virtual display with user instructions.
!-
ret_status = SMG$PUT_LINE(display3_id, instr_0, advance,,, wrap)
IF (ret_status AND 1%) = 0% THEN
   CALL LIB$STOP(ret_status BY VALUE)
END IF
ret_status = SMG$PUT_LINE(display3_id, instr_1, advance,,, wrap)
IF (ret_status AND 1%) = 0% THEN
   CALL LIB$STOP(ret_status BY VALUE)
END IF
ret_status = SMG$PUT_LINE(display3_id, instr_2, advance,,, wrap)
IF (ret_status AND 1%) = 0% THEN
   CALL LIB$STOP(ret_status BY VALUE)
END IF

!+
!Paste the virtual displays to the screen.
!-
ret_status = SMG$PASTE_VIRTUAL_DISPLAY(display3_id, pb_id, &
        2 BY REF, 4 BY REF)
IF (ret_status AND 1%) = 0% THEN
   CALL LIB$STOP(ret_status BY VALUE)
END IF
ret_status = SMG$PASTE_VIRTUAL_DISPLAY(display_id, pb_id, &
        8 BY REF, 4 BY REF)
IF (ret_status AND 1%) = 0% THEN
   CALL LIB$STOP(ret_status BY VALUE)
END IF
ret_status = SMG$PASTE_VIRTUAL_DISPLAY(display2_id, pb_id, &
        18 BY REF, 4 BY REF)
IF (ret_status AND 1%) = 0% THEN
   CALL LIB$STOP(ret_status BY VALUE)
END IF
!+
!Enable the trapping of unsolicited input. GET_INPUT is the
!AST procedure that is called when unsolicited input is
!received. This AST has one parameter, passed as null.
!-
ret_status = SMG$ENABLE_UNSOLICITED_INPUT(pb_id, GET_INPUT,)
IF (ret_status AND 1%) = 0% THEN
   CALL LIB$STOP(ret_status BY VALUE)
END IF

!+
!Enable the trapping of broadcast messages. GET_MSG is the
!AST which is called when broadcast messages are received.
!This AST outputs the trapped message into the MESSAGES display.
!-
ret_status = SMG$SET_BROADCAST_TRAPPING(pb_id, GET_MSG)
IF (ret_status AND 1%) = 0% THEN
   CALL LIB$STOP(ret_status BY VALUE)
END IF

!+
!This loop continually executes until done_flag is set to 1.
```

```
        !Done_flag is set to 1 when the user input is a CTRL/Z.
        !If done_flag is 1, delete the pasteboard and exit the program.
        !-
       Infinite_loop:
        IF done_flag = 0 THEN
           GOTO infinite_loop
        ELSE
           ret_status = SMG$DELETE_PASTEBOARD (pb_id)
           GOTO all_done
        END IF
      All_done:
        END

20      !+
        !Start of AST routine GET_INPUT. This AST is called whenever there
        !is unsolicited input. The unsolicited input is displayed in the
        !INPUT virtual display, and if this input is not CTRL/Z, a subprocess
        !is spawned and the input command is executed. While this spawned
        !subprocess is executing, broadcast and unsolicited input trapping
        !is disabled.
        !-
        SUB GET_INPUT (paste_id, param, null_1, null_2, null_3, null_4)
        MAP (params) LONG disp_info(2), LONG keyboard_info(4), LONG done_flag
        DECLARE LONG z_status, status2, keybd_id, keybd_tab_id, disp_id, &
                     pastebd, new_display
        DECLARE WORD msg2_len
        DECLARE STRING msg2
        DECLARE LONG CONSTANT next_line = 1
        DECLARE LONG CONSTANT wrap_flag = 1

        EXTERNAL LONG FUNCTION SMG$READ_COMPOSED_LINE (LONG, LONG, STRING, &
                                  STRING, WORD, LONG)
        EXTERNAL LONG FUNCTION SMG$SET_BROADCAST_TRAPPING (LONG, LONG)
        EXTERNAL LONG FUNCTION SMG$ENABLE_UNSOLICITED_INPUT (LONG, LONG, &
                                  LONG)
        EXTERNAL LONG FUNCTION SMG$DISABLE_UNSOLICITED_INPUT (LONG)
        EXTERNAL LONG FUNCTION SMG$DISABLE_BROADCAST_TRAPPING (LONG)
        EXTERNAL LONG FUNCTION SMG$SAVE_PHYSICAL_SCREEN (LONG, LONG)
        EXTERNAL LONG FUNCTION SMG$RESTORE_PHYSICAL_SCREEN (LONG, LONG)
        EXTERNAL SUB LIB$SPAWN (STRING)
        EXTERNAL SUB LIB$STOP (LONG BY VALUE)
        EXTERNAL INTEGER       GET_MSG
        EXTERNAL INTEGER       GET_INPUT
        EXTERNAL LONG CONSTANT SMG$_EOF
        EXTERNAL LONG CONSTANT SS$_NORMAL

        !+
        !Assign to the local variables the values that were stored from
        !the main program using the MAP area.
        !-
        keybd_id = keyboard_info(0)
        keybd_tab_id = keyboard_info(1)
        disp_id = keyboard_info(2)
        pastebd = keyboard_info(3)

        !+
        !SMG$ENABLE_UNSOLICITED_INPUT does not read the input, it simply
        !signals the specified AST when there is unsolicited input present.
        !You must use SMG$READ_COMPOSED_LINE to actually read the input.
        !
        !At this time, we check to see if the unsolicited input was a CTRL/Z.
        !If so, we skip over the program lines that spawn the subprocess and
        !get ready to exit the program.
        !-
        status2 = SMG$READ_COMPOSED_LINE (keybd_id, keybd_tab_id, msg2,, &
                    msg2_len, disp_id)
        IF (status2 = SMG$_EOF) THEN
           GOTO Control_Z
        END IF
        IF (status2 AND 1%) = 0% THEN
           CALL LIB$STOP (status2 BY VALUE)
```

```
    END IF
    !+
    !In order to spawn a subprocess, we must first disable
    !unsolicited input trapping and broadcast trapping.
    !-
    status2 = SMG$DISABLE_UNSOLICITED_INPUT (pastebd)
    IF (status2 AND 1%) = 0% THEN
       CALL LIB$STOP (status2 BY VALUE)
    END IF
    status2 = SMG$DISABLE_BROADCAST_TRAPPING (pastebd)
    IF (status2 AND 1%) = 0% THEN
       CALL LIB$STOP (status2 BY VALUE)
    END IF
    !+
    !Save the current screen so that it will not be destroyed when
    !the subprocess is executing.
    !-
    status2 = SMG$SAVE_PHYSICAL_SCREEN (pastebd, new_display)
    IF (status2 AND 1%) = 0% THEN
       CALL LIB$STOP (status2 BY VALUE)
    END IF
    !+
    !Call LIB$SPAWN to create the subprocess, and pass the unsolicited
    !input as the command line.
    !-
    CALL LIB$SPAWN (msg2)
    !+
    !Restore the saved screen image.
    !-
    status2 = SMG$RESTORE_PHYSICAL_SCREEN (pastebd, new_display)
    IF (status2 AND 1%) = 0% THEN
       CALL LIB$STOP (status2 BY VALUE)
    END IF
    !+
    !Reenable broadcast trapping and unsolicited input trapping.
    !-
    status2 = SMG$SET_BROADCAST_TRAPPING (pastebd, GET_MSG)
    IF (status2 AND 1%) = 0% THEN
       CALL LIB$STOP (status2 BY VALUE)
    END IF
    status2 = SMG$ENABLE_UNSOLICITED_INPUT (pastebd, GET_INPUT,)
    IF (status2 AND 1%) = 0% THEN
       CALL LIB$STOP (status2 BY VALUE)
    END IF
    !+
    !Skip the steps which are preformed if the unsolicited input
    !was a CTRL/Z.
    !-
    GOTO Out_of_sub
Control_Z:
    !+
    !We have to disable unsolicited input and broadcast trapping
    !before we can leave the program.
    !-
    status2 = SMG$DISABLE_UNSOLICITED_INPUT (pastebd)
    IF (status2 AND 1%) = 0% THEN
       CALL LIB$STOP (status2 BY VALUE)
    END IF
    status2 = SMG$DISABLE_BROADCAST_TRAPPING (pastebd)
    IF (status2 AND 1%) = 0% THEN
       CALL LIB$STOP (status2 BY VALUE)
    END IF
    !+
    !Set the done_flag to 1 so that the main program knows we have
    !to exit.
    !-
```

```
        done_flag = 1
      Out_of_sub:
        END SUB

30      !+
        !Start of AST routine GET_MSG. This AST is called whenever there
        !is a broadcast message. This routine prints the message in the
        !MESSAGES virtual display.
        !-
        SUB GET_MSG (paste_id, nl_1, nl_2, nl_3, nl_4)
        DECLARE LONG status1, pasteboard, second_disp
        DECLARE WORD msg_len
        DECLARE STRING msg
        DECLARE LONG CONSTANT forward = 1
        DECLARE LONG CONSTANT wrp_flag = 1
        MAP (params) LONG disp_info(2), LONG keyboard_info(4)
        EXTERNAL LONG FUNCTION SMG$PUT_LINE (LONG, STRING, LONG, LONG, &
                                LONG, LONG)
        EXTERNAL LONG FUNCTION SMG$GET_BROADCAST_MESSAGE (LONG, STRING, &
                                WORD)
        EXTERNAL SUB LIB$STOP (LONG BY VALUE)
        EXTERNAL LONG CONSTANT SMG$_NO_MORMSG
        !+
        !Assign values to the local variables according to the values stored
        !in the MAP area.
        !-
        pasteboard = disp_info(0)
        second_disp = disp_info(1)
        !+
        !Print the trapped message in the MESSAGES display. If there are no
        !more messages, go back to the infinite loop in the main program.
        !-
        WHILE 1
            status1 = SMG$GET_BROADCAST_MESSAGE (pasteboard, msg, msg_len)
            IF (status1 = SMG$_NO_MORMSG) THEN
               GOTO Exitloop
            END IF
            IF (status1 AND 1%) = 0% THEN
               CALL LIB$STOP (status1 BY VALUE)
            END IF
            status1 = SMG$PUT_LINE (second_disp, msg, forward,,, wrp_flag)
            IF (status1 AND 1%) = 0% THEN
               CALL LIB$STOP (status1 BY VALUE)
            END IF
        NEXT
      Exitloop:
         END SUB
```

Note that this program must be linked with the file SMGDEF.MAR. You build this file using the following commands.

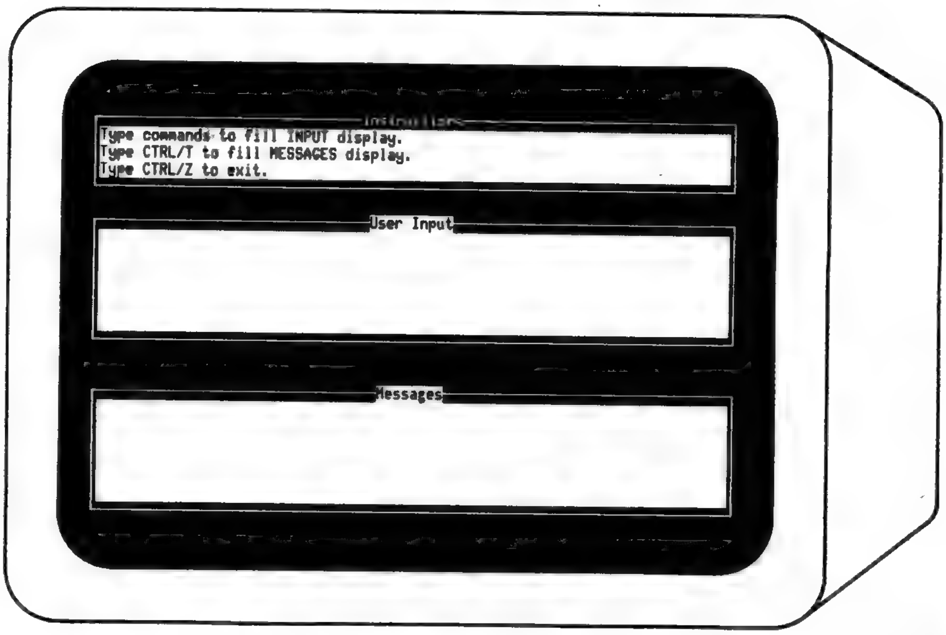
```
$ CREATE SMGDEF.MAR
        .TITLE  SMGDEF - Define SMG$ constants
        .Ident  /1-000/
        $SMGDEF GLOBAL
        .END
$ MACRO SMGDEF
```

To run the example program, use the following commands.

```
$ BASIC TRAPPING
$ LINK TRAPPING,SMGDEF
$ RUN TRAPPING
```

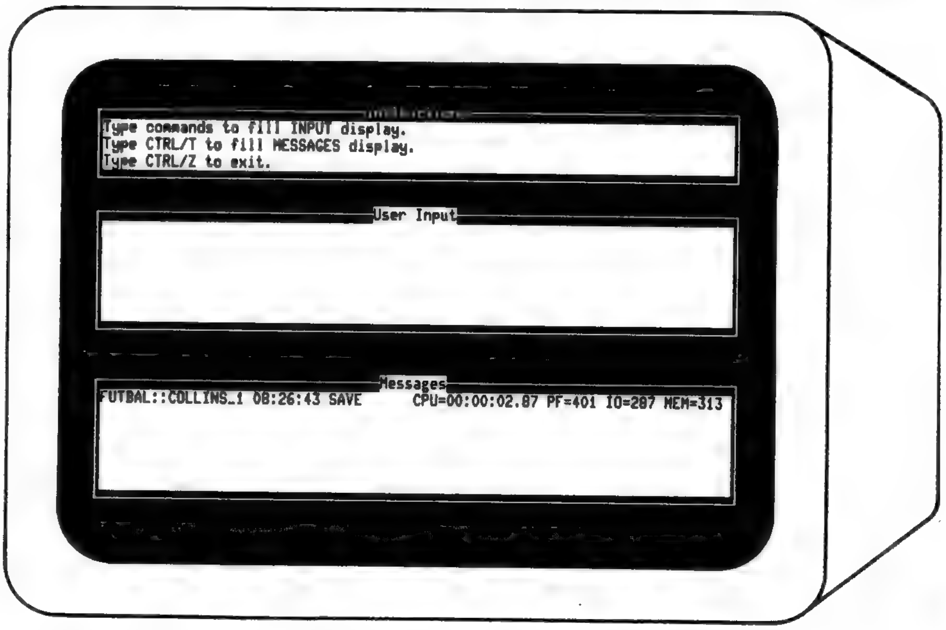
The output for this program is illustrated in the following figures. In Figure RTL–14.1, the program is waiting for either unsolicited input or broadcast messages.

Figure RTL–14.1 Output Generated Before Any Input or Messages Are Trapped



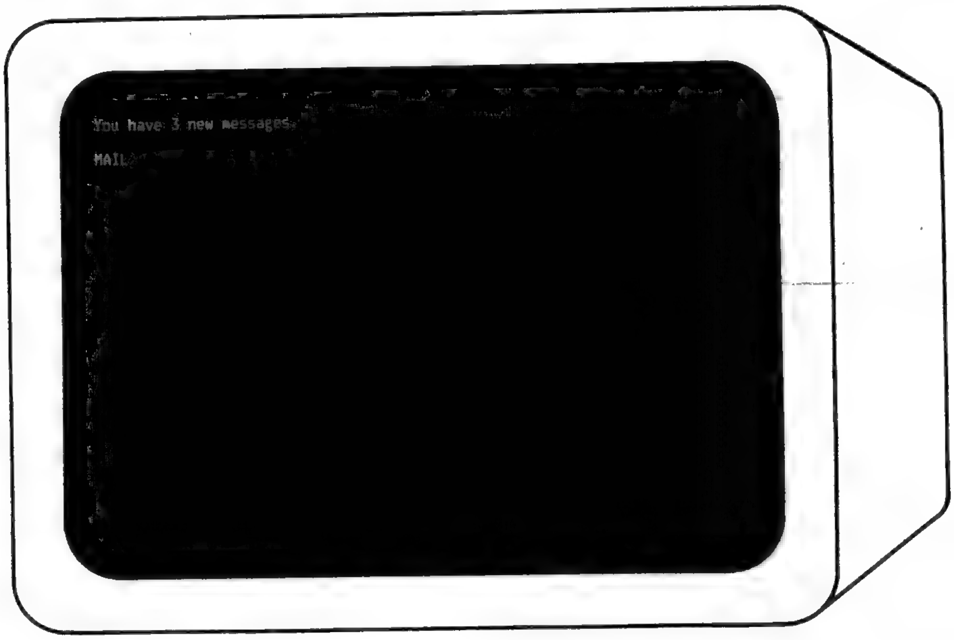
The output generated after the user types a CTRL/T is shown in Figure RTL–14.2.

Figure RTL–14.2 Output Generated After a Broadcast Message Is Trapped



If the user types a command, that command is displayed in the INPUT display, and a subprocess is spawned. The output generated after the user types the command MAIL is shown in Figure RTL–14.3.

Figure RTL–14.3 Output Generated After a Call to LIB$SPAWN



ZK-4807-85

Once the subprocess completes execution, control is returned to the main process. At this point, the screen is repainted and the program continues to wait for broadcast messages or unsolicited input. The user must type a CTRL/Z to exit the program.

SMG$DISABLE_UNSOLICITED_INPUT Disable Unsolicited Input

SMG$DISABLE_UNSOLICITED_INPUT disables the invocation of AST routines for unsolicited input.

FORMAT

SMG$DISABLE_UNSOLICITED_INPUT *pasteboard-id*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENT

pasteboard-id

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Specifies the keyboard (associated with the specified pasteboard) for which unsolicited input is being disabled. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier. **Pasteboard-id** is returned by SMG$CREATE_PASTEBOARD.

DESCRIPTION

SMG$DISABLE_UNSOLICITED_INPUT disables unsolicited input ASTs for the specified pasteboard. SMG$DISABLE_UNSOLICITED_INPUT deassigns the mailbox set with SMG$ENABLE_UNSOLICITED_INPUT, resets the terminal characteristics, and therefore allows the user to SPAWN a subprocess. This routine must be used to disable any unsolicited input trapping enabled with the SMG$ENABLE_UNSOLICITED_INPUT routine.

Note that if both unsolicited input trapping and the trapping of broadcast messages are enabled, then both SMG$DISABLE_UNSOLICITED_INPUT and SMG$DISABLE_BROADCAST_TRAPPING must be invoked in order to deassign the mailbox.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_INVPAS_ID	Invalid **pasteboard-id**.

Any condition values returned by $QIOW.

EXAMPLE

For an example of using SMG$DISABLE_UNSOLICITED_INPUT, see the example for the routine SMG$DISABLE_BROADCAST_TRAPPING.

SMG$DRAW_LINE—Draw A Line

SMG$DRAW_LINE draws a horizontal or vertical line.

FORMAT

SMG$DRAW_LINE *display-id ,start-row ,start-column ,end-row ,end-column [,rendition-set] [,rendition-complement]*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

display-id

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Specifies the virtual display on which the line is to be drawn. The **display-id** argument is the address of an unsigned longword that contains the display identifier. **Display-id** is returned by SMG$CREATE_VIRTUAL_DISPLAY.

start-row

VMS Usage:	**longword_signed**
type:	**longword integer (signed)**
access:	**read only**
mechanism:	**by reference**

Specifies the row at which to begin drawing the line. The **start-row** argument is the address of a signed longword integer that contains the row number at which to begin drawing the line.

start-column

VMS Usage:	**longword_signed**
type:	**longword integer (signed)**
access:	**read only**
mechanism:	**by reference**

Specifies the column at which to begin drawing the line. The **start-column** argument is the address of a signed longword integer that contains the column number at which to begin drawing the line.

end-row

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the row at which the drawn line ends. The **end-row** argument is the address of a signed longword integer that contains the row number at which the drawn line ends.

end-column

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the column at which the drawn line ends. The **end-column** argument is the address of a signed longword integer that contains the column number at which the drawn line ends.

rendition-set

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

A mask which denotes video attributes for the drawn line. The **rendition-set** argument is the address of an unsigned longword that contains a video attributes mask. Each bit attribute in this argument causes the corresponding attribute to be set in the display.

Video attributes which can be manipulated in this manner are as follows:

SMG$M_BLINK	Displays blinking characters
SMG$M_BOLD	Displays characters in higher-than-normal intensity
SMG$M_REVERSE	Displays characters in reverse video, that is, using the opposite default rendition of the virtual display
SMG$M_UNDERLINE	Displays underlined characters

If the same bit is set in both the **rendition-set** and **rendition-complement** arguments, the Screen Management Facility applies the **rendition-set** attribute followed by the **rendition-complement** attribute. Using these two arguments, the caller can exercise independent control over each attribute in a single call.

rendition-complement

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

A mask which denotes video attributes for the line drawn. The **rendition-complement** argument is the address of an unsigned longword that contains a video attributes mask. Each bit attribute in this argument causes the corresponding attribute to be set in the display. Video attributes which can be manipulated in this manner are the same as those for the **rendition-set** argument.

The following table shows the action taken by the Screen Management Facility for various combinations of **rendition-set** and **rendition-complement** attributes:

Set	Complement	Action
0	0	Attribute unchanged
1	0	Attribute on
0	1	Attribute set to complement of default setting
1	1	Attribute off

DESCRIPTION

SMG$DRAW_LINE draws a line from a specified starting row and column to a specified ending row and column. You can draw only horizontal or vertical lines. The characters used to draw the line depend on the type of terminal. The virtual cursor position does not change.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVCOL	Invalid column number. The specified column is outside the virtual display.
SMG$_INVROW	Invalid row number. The specified row is outside the virtual display
SMG$_DIALINNOT	Diagonal line not allowed.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_INVDIS_ID	Invalid **display-id**.

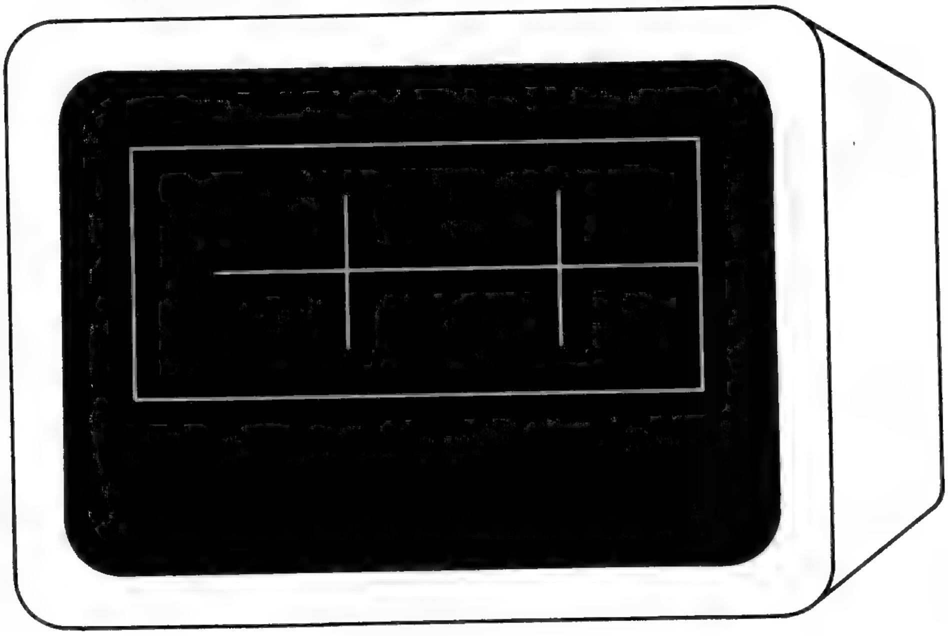
EXAMPLE

```
C+
C This FORTRAN example program demonstrates the use of SMG$DRAW_LINE.
C-
        INTEGER SMG$CREATE_VIRTUAL_DISPLAY, SMG$CREATE_PASTEBOARD
        INTEGER SMG$PASTE_VIRTUAL_DISPLAY, SMG$DRAW_LINE
        INTEGER DISPLAY1, PASTE1, ROWS, COLUMNS, BORDER, STATUS
C+
C First, create the virtual display using SMG$CREATE_VIRTUAL_DISPLAY.
C To give it a border, set BORDER = 1. No border would be BORDER = 0.
C-
        ROWS = 7
        COLUMNS = 50
        BORDER = 1
        STATUS = SMG$CREATE_VIRTUAL_DISPLAY
     1                            (ROWS, COLUMNS, DISPLAY1, BORDER)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Call SMG$CREATE_PASTEBOARD to create the pasteboard.
C-
        STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Draw a verticle line using SMG$DRAW_LINE.
C Start at row 2, column 20. End at row 6.
C-
        STATUS = SMG$DRAW_LINE (DISPLAY1, 2, 20, 6, 20)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
```

```
C+
C Now, use SMG$DRAW_LINE to draw a verticle line.
C Start at row 6, column 40. End at row 2.
C This is similar to the line drawn above, but we are drawing the
C line in the reverse direction.
C-
        STATUS = SMG$DRAW_LINE (DISPLAY1, 6, 40, 2, 40)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Draw a horizontal line now, again calling SMG$DRAW_LINE.
C Start at row 4, column 8. End at column 50.
C-
        STATUS = SMG$DRAW_LINE (DISPLAY1, 4, 8, 4, 50)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Paste the virtual display using SMG$PASTE_VIRTUAL_DISPLAY.
C-
        STATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DISPLAY1, PASTE1, 4, 15)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        END
```

The output generated by this FORTRAN example is shown in Figure RTL-15.

Figure RTL-15 Output Generated by FORTRAN Program Calling SMG$DRAW_LINE



ZK-4110-85

SMG$DRAW_RECTANGLE—Draw a Rectangle

SMG$DRAW_RECTANGLE draws a rectangle.

FORMAT

SMG$DRAW_RECTANGLE *display-id ,top-left-row ,top-left-column ,bottom-right-row ,bottom-right-column [,rendition-set] [,rendition-complement]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the virtual display on which the rectangle is to be drawn. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

Display-id is returned by SMG$CREATE_VIRTUAL_DISPLAY.

top-left-row

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the row number of the top left-hand corner of the rectangle. The **top-left-row** argument is the address of a signed longword integer that contains the row number of the top left-hand corner of the rectangle.

top-left-column

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the column number of the top left-hand corner of the rectangle. The **top-left-column** argument is the address of a signed longword integer that contains the column number of the top left-hand corner of the rectangle.

bottom-right-row

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the row number of the bottom right-hand corner of the rectangle. The **bottom-right-row** argument is the address of a signed longword integer that contains the row number of the bottom right-hand corner of the rectangle.

bottom-right-column

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the column number of the bottom right-hand corner of the rectangle. The **bottom-right-column** argument is the address of a signed longword integer that contains the column number of the bottom right-hand corner of the rectangle.

rendition-set

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Mask that denotes video attributes for the drawn line. The **rendition-set** argument is the address of an unsigned longword that contains a video attributes mask. Each bit attribute in this argument causes the corresponding attribute to be set in the display.

Video attributes that can be manipulated in this manner are as follows:

SMG$M_BLINK	Displays blinking characters
SMG$M_BOLD	Displays characters in higher-than-normal intensity
SMG$M_REVERSE	Displays characters in reverse video, that is, using the opposite default rendition of the virtual display
SMG$M_UNDERLINE	Displays underlined characters

If the same bit is set in both the **rendition-set** and **rendition-complement** arguments, the Screen Management Facility applies the **rendition-set** attribute followed by the **rendition-complement** attribute. Using these two arguments, the caller can exercise independent control over each attribute in a single call.

rendition-complement

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Mask that denotes video attributes for the line drawn. The **rendition-complement** argument is the address of an unsigned longword that contains a video attributes mask. Each bit attribute in this argument causes the corresponding attribute to be set in the display. Video attributes that can be manipulated in this manner are the same as for the **rendition-set** argument.

The following table shows the action taken by the Screen Management Facility for various combinations of **rendition-set** and **rendition-complement** attributes.

Set	Complement	Action
0	0	Attribute unchanged
1	0	Attribute on
0	1	Attribute set to complement of default setting
1	1	Attribute off

DESCRIPTION

SMG$DRAW_RECTANGLE draws a rectangle in a virtual display, given the position of the upper left-hand corner and the lower right-hand corner. The characters used to draw the lines making up the rectangle depend on the type of terminal. The virtual cursor position does not change.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVCOL	Invalid column number. The specified column is outside the virtual display.
SMG$_INVROW	Invalid row number. The specified row is outside the virtual display.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_INVDIS_ID	Invalid **display-id**.

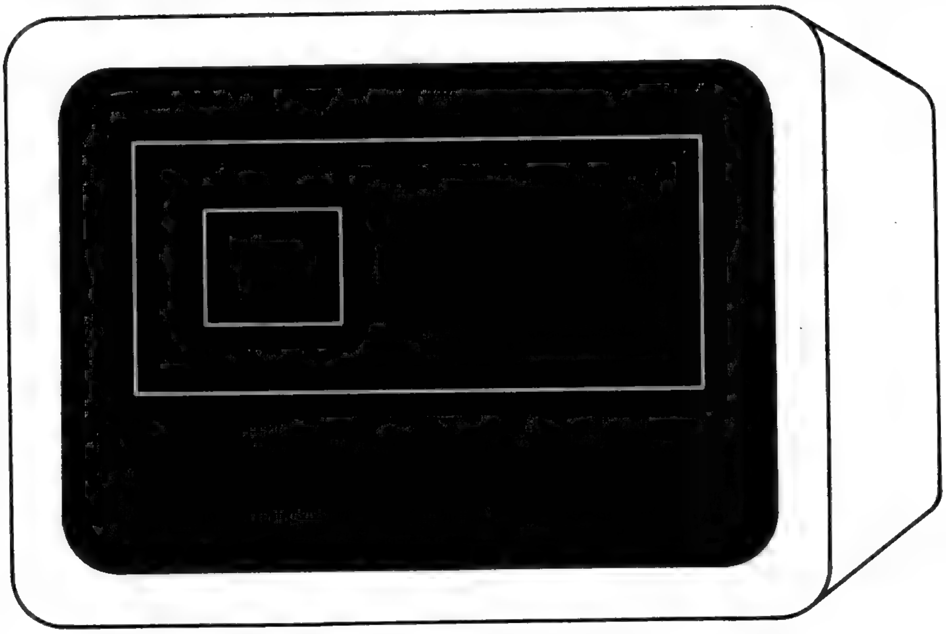
EXAMPLE

```
C+
C This FORTRAN example program demonstrates the use of
C SMG$DRAW_RECTANGLE.
C
C This routine creates a virtual display and uses SMG$DRAW_RECTANGLE
C to draw a rectangle inside the bordered virtual display.
C-
C+
C Include the SMG definitions. In particular, we want SMG$M_BORDER.
C-

        INCLUDE '($SMGDEF)'
        INTEGER SMG$CREATE_VIRTUAL_DISPLAY, SMG$CREATE_PASTEBOARD
        INTEGER SMG$PASTE_VIRTUAL_DISPLAY, SMG$DRAW_RECTANGLE
        INTEGER DISPLAY1, PASTE1, ROWS, COLUMNS, STATUS
C+
C Create a virtual display with a border by calling
C SMG$CREATE_VIRTUAL_DISPLAY.
C-

        ROWS = 7
        COLUMNS = 50
        STATUS = SMG$CREATE_VIRTUAL_DISPLAY
     1                  (ROWS, COLUMNS, DISPLAY1, SMG$M_BORDER)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Use SMG$CREATE_PASTEBOARD to create the pasteboard.
C-

        STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
```

```
C+
C Using SMG$DRAW_RECTANGLE, draw a rectangle inside the bordered region.
C-
        STATUS = SMG$DRAW_RECTANGLE (DISPLAY1, 2, 10, 6, 20)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Paste the virtual display by calling SMG$PASTE_VIRTUAL_DISPLAY.
C-
        STATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DISPLAY1, PASTE1, 4, 15)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        END
```

The output generated by this FORTRAN example is shown in Figure RTL–16.

Figure RTL–16 Output Generated by FORTRAN Program Calling SMG$DRAW_RECTANGLE



SMG$ENABLE_UNSOLICITED_INPUT Enable Unsolicited Input

SMG$ENABLE_UNSOLICITED_INPUT detects unsolicited input and calls an AST routine in response.

FORMAT

SMG$ENABLE_UNSOLICITED_INPUT *pasteboard-id ,AST-routine [,AST-argument]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

pasteboard-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the pasteboard for which unsolicited input is being enabled. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier. **Pasteboard-id** is returned by SMG$CREATE_VIRTUAL_PASTEBOARD.

AST-routine

VMS Usage: **ast_procedure**
type: **procedure entry mask**
access: **read only**
mechanism: **by reference**

The address of an AST routine to be called upon receipt of unsolicited input at the terminal. The **AST-routine** argument is the address of the routine's procedure entry mask, that is, the address of the routine itself.

AST-argument

VMS Usage: **user_arg**
type: **longword**
access: **read only**
mechanism: **by value**

A value to be passed to the AST routine. The **AST-argument** argument contains the value to be passed to the AST routine.

DESCRIPTION

SMG$ENABLE_UNSOLICITED_INPUT detects the presence of unsolicited input and calls the AST routine with six arguments: the **pasteboard-id**, the **AST-argument**, R0, R1, PC, and PSL.

Pasteboard-id
AST-arg
R0
R1
PC
PSL

ZK-4802-85

Note that this routine does not read any input characters; it merely calls an AST routine to "notify" the application that it should issue a read operation with SMG$READ_COMPOSED_LINE, SMG$READ_KEYSTROKE, SMG$READ_STRING or SMG$READ_VERIFY. It is up to you to read the unsolicited input.

SMG$ENABLE_UNSOLICITED_INPUT establishes a mailbox that receives messages when terminal-related events occur that require the attention of the user image. This mailbox carries status messages, not terminal data, from the driver to the user program. This status message is sent to the mailbox when there is unsolicited data in the type-ahead buffer. In this case, the user process enters into a dialogue with the terminal after an unsolicited data message arrives. Once this dialogue is complete, the Screen Management Facility reenables the unsolicited data message function on the last I/O exchange. Only one message is sent between read operations.

For more information on terminal/mailbox interaction, see the *VAX/VMS I/O User's Reference Manual: Part I.*

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_INVPBD_ID	Invalid **pasteboard-id**.

Any condition values returned by $QIOW.

EXAMPLE

For an example of using SMG$ENABLE_UNSOLICITED_INPUT, see the example for the routine SMG$DISABLE_BROADCAST_TRAPPING.

SMG$END_DISPLAY_UPDATE End Display Update

SMG$END_DISPLAY_UPDATE ends update batching for a virtual display.

FORMAT

SMG$END_DISPLAY_UPDATE *display-id*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENT

display-id

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Specifies the virtual display to be affected. The **display-id** argument is the address of a longword that contains the display identifier.

Display-id is returned by SMG$CREATE_VIRTUAL_DISPLAY.

DESCRIPTION

SMG$END_DISPLAY_UPDATE and SMG$BEGIN_DISPLAY_UPDATE work together to control the batching of output operations on a given virtual display. Each call to SMG$BEGIN_DISPLAY_UPDATE increments a "batch count," while each call to SMG$END_DISPLAY_UPDATE decrements this count. When the batch count reaches 0, the virtual display is written to the screen.

Calling SMG$END_DISPLAY_UPDATE when the batch count is zero is a valid operation; therefore a success status is returned.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_BATWASOFF	Successful completion. Note that batching was already off.
SMG$_BATSTIPRO	Successful completion. Note that batching is still in progress.
SMG$_INVDIS_ID	Invalid **display-id**.
SMG$_WRONUMARG	Wrong number of arguments.

SMG$END_PASTEBOARD_UPDATE End Pasteboard Update

SMG$END_PASTEBOARD_UPDATE ends update batching for a pasteboard.

FORMAT

SMG$END_PASTEBOARD_UPDATE *pasteboard-id*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENT

pasteboard-id

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Specifies the pasteboard on which the batch count is to be decremented. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier. **Pasteboard-id** is returned by SMG$CREATE_PASTEBOARD. If the batch count reaches 0, all buffered output for the specified pasteboard is written out.

DESCRIPTION

SMG$END_PASTEBOARD_UPDATE and SMG$BEGIN_PASTEBOARD_UPDATE work together to control the batching of output operations on a given pasteboard. Each call to SMG$BEGIN_PASTEBOARD_UPDATE increments a "batch count," while each call to SMG$END_PASTEBOARD_UPDATE decrements this count. When the batch count reaches 0, the pasteboard is written to the screen.

Calling SMG$END_PASTEBOARD_UPDATE when the batch count is 0 is a valid operation; a success status is returned.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_BATWASOFF	Successful completion. Note that batching was already off.
SMG$_BATSTIPRO	Successful completion. Note that batching is still in progress.
SMG$_INVDIS_ID	Invalid **display-id**.
SMG$_WRONUMARG	Wrong number of arguments.

SMG$ERASE_CHARS—Erase Characters

SMG$ERASE_CHARS erases characters in a virtual display by replacing them with blanks.

FORMAT

SMG$ERASE_CHARS *display-id ,number-of-chars ,row-number ,column-number*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the virtual display from which characters will be erased. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

Display-id is returned by SMG$CREATE_VIRTUAL_DISPLAY.

number-of-chars

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the number of characters to be replaced with blanks. The **number-of-chars** argument is the address of a signed longword integer that contains the number of characters to be replaced with blanks.

row-number

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the row on which the erase operation begins. The **row-number** argument is the address of a signed longword integer that contains the number of the row at which the erasure is to begin.

column-number

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the column on which the erase operation begins. The **column-number** argument is the address of a signed longword integer that contains the number of the column at which the erasure is to begin.

DESCRIPTION

SMG$ERASE_CHARS erases characters in a virtual display by replacing them with blanks. The remaining text in the display is not moved. An erase operation is limited to the specified line. If **number-of-chars** is greater than the number of characters remaining in the line, all characters from the specified starting position to the end of the line are erased. This routine leaves the virtual cursor at the position of the first character erased.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVDIS_ID	Invalid **display-id**.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_INVROW	Invalid row.
SMG$_INVCOL	Invalid column.

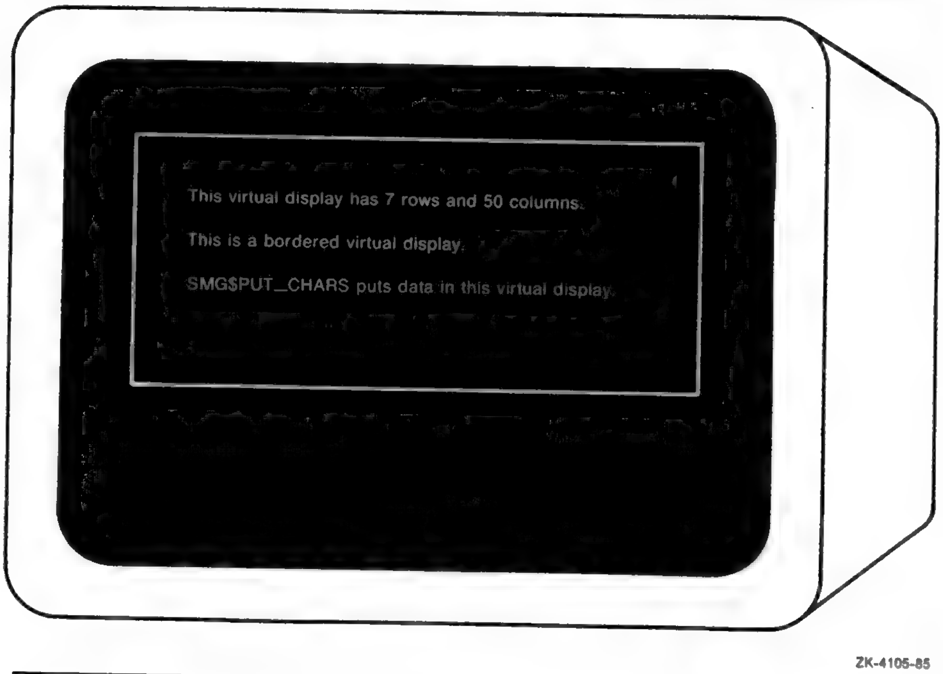
EXAMPLE

```
C+
C This FORTRAN example demonstrates the use of SMG$ERASE_CHARS.
C
C Include the SMG definitions. In particular, we want SMG$M_BORDER.
C-
        INCLUDE '($SMGDEF)'
        INTEGER SMG$CREATE_VIRTUAL_DISPLAY, SMG$CREATE_PASTEBOARD
        INTEGER SMG$PASTE_VIRTUAL_DISPLAY,  SMG$PUT_CHARS, SMG$ERASE_CHARS
        INTEGER DISPLAY1, PASTE1, ROWS, COLUMNS

C+
C Create a virtual display with a border by calling
C SMG$CREATE_VIRTUAL_DISPLAY.
C-
        ROWS = 7
        COLUMNS = 50
        ISTATUS = SMG$CREATE_VIRTUAL_DISPLAY
     1          (ROWS, COLUMNS, DISPLAY1, SMG$M_BORDER)

C+
C Call SMG$CREATE_PASTEBOARD to create the pasteboard.
C-
        ISTATUS = SMG$CREATE_PASTEBOARD (PASTE1)

C+
C Using SMG$PUT_CHARS, put data in the virtual display.
C-
        ISTATUS = SMG$PUT_CHARS ( DISPLAY1,
     1  ' This virtual display has 7 rows and 50 columns.', 2, 1)
        ISTATUS = SMG$PUT_CHARS ( DISPLAY1,
     1          ' This is a bordered virtual display.', 4, 1)
        ISTATUS = SMG$PUT_CHARS ( DISPLAY1,
     1  ' SMG$PUT_CHARS puts data in this virtual display.', 6, 1)

C+
```

```
C Call SMG$PASTE_VIRTUAL_DISPLAY to paste the virtual display.
C-
        ISTATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DISPLAY1, PASTE1, 4, 15)
C+
C Erase 4 characters on row 4 starting from character (column) 14 by
C calling SMG$ERASE_CHARS.  This will remove the characters "rder"
C from the word "bordered".
C-
        ISTATUS = SMG$ERASE_CHARS ( DISPLAY1, 4, 4, 14)
        END
```

The initial output generated by this FORTRAN example program is shown in Figure RTL–17.

Figure RTL–17 Output Before the Call to SMG$ERASE_CHARS



The output generated after the call to SMG$ERASE_CHARS is shown in Figure RTL–18.

Figure RTL-18 Output After the Call to SMG$ERASE_CHARS

ZK-4113-85

SMG$ERASE_DISPLAY—Erase Virtual Display

SMG$ERASE_DISPLAY erases all or part of a virtual display by replacing text characters with blanks.

FORMAT

SMG$ERASE_DISPLAY *display-id [,start-row] [,start-column] [,end-row] [,end-column]*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

display-id

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Specifies the virtual display to be erased. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

Display-id is returned by SMG$CREATE_VIRTUAL_DISPLAY.

start-row

VMS Usage:	**longword_signed**
type:	**longword integer (signed)**
access:	**read only**
mechanism:	**by reference**

Specifies the row at which the erase operation begins. The **start-row** argument is the address of a signed longword integer that contains the number of the row at which the erasure begins.

If the **start-row** argument is not specified, **start-column** is also ignored and the entire virtual display is erased.

start-column

VMS Usage:	**longword_signed**
type:	**longword integer (signed)**
access:	**read only**
mechanism:	**by reference**

Specifies the column at which the erase operation begins. The **start-column** argument is the address of a signed longword integer that contains the number of the column at which the erasure begins.

If the **start-column** argument is not specified, **start-row** is also ignored and the entire virtual display is erased.

end-row

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the row at which the erase operation ends, that is, the last row to be erased. The **end-row** argument is the address of a signed longword integer that contains the number of the last row to be erased.

If the **end-row** argument is not specified, **end-column** is also ignored and all remaining rows in the display are erased.

end-column

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the column at which the erase operation ends, that is, the last column to be erased. The **end-column** argument is the address of a signed longword integer that contains the number of the last column to be erased.

If the **end-column** argument is not specified, **end-row** is also ignored and all remaining columns in the display are erased.

DESCRIPTION

SMG$ERASE_DISPLAY causes all or part of a virtual display to be erased by replacing text characters with blanks. If omitted, the starting positions default to 1,1. The ending positions default to the last row or column in the display. Thus, to erase the entire virtual display, you need only pass the **display-id**. The cursor position is the first free position after the erased portion. If the entire display is erased, the virtual cursor is left at position 1,1.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVDIS_ID	Invalid **display-id**.
SMG$_INVCOL	Invalid column number. The specified column is outside the virtual display.
SMG$_INVROW	Invalid row number. The specified row is outside the virtual display.
SMG$_WRONUMARG	Wrong number of arguments.

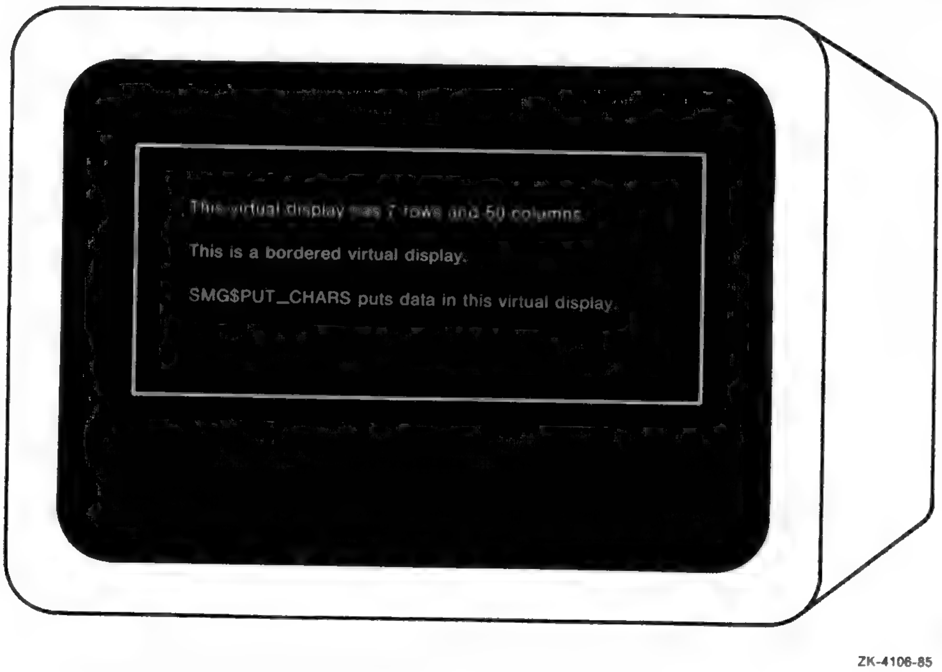
EXAMPLE

```
C+
C This FORTRAN example program illustrates the use of SMG$ERASE_DISPLAY.
C-
        INTEGER SMG$CREATE_VIRTUAL_DISPLAY, SMG$CREATE_PASTEBOARD
        INTEGER SMG$PASTE_VIRTUAL_DISPLAY,  SMG$PUT_CHARS, SMG$ERASE_DISPLAY
        INTEGER DISPLAY1, PASTE1, ROWS, COLUMNS, BORDER

C+
C Call SMG$CREATE_VIRTUAL_DISPLAY to create the virtual
C display. To give it a border, set BORDER = 1.
C No border would be BORDER = 0.
C-
        ROWS = 7
        COLUMNS = 50
```

```
        BORDER = 1
        ISTATUS = SMG$CREATE_VIRTUAL_DISPLAY
     1                         (ROWS, COLUMNS, DISPLAY1, BORDER)
        IF (.NOT. ISTATUS) WRITE (6, 900) 'SMG$CREATE_VIRTUAL_DISPLAY', ISTATUS
C+
C Using SMG$CREATE_PASTEBOARD, create the pasteboard.
C-
        ISTATUS = SMG$CREATE_PASTEBOARD (PASTE1)
        IF (.NOT. ISTATUS) WRITE (6, 900) 'SMG$CREATE_PASTEBOARD', ISTATUS
C+
C Call SMG$PUT_CHARS to put data in the virtual display.
C-
        ISTATUS = SMG$PUT_CHARS ( DISPLAY1,
     1          ' This virtual display has 7 rows and 50 columns.', 2, 1)
        ISTATUS = SMG$PUT_CHARS ( DISPLAY1,
     1          ' This is a bordered virtual display.', 4, 1)
        ISTATUS = SMG$PUT_CHARS ( DISPLAY1,
     1          ' SMG$PUT_CHARS puts data in this virtual display.', 6, 1)
C+
C Paste the virtual display by calling SMG$PASTE_VIRTUAL_DISPLAY.
C-
        ISTATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DISPLAY1, PASTE1, 4, 15)
900     FORMAT (' Routine ', A, ' returned a status of ', Z8)
C+
C Call SMG$ERASE_DISPLAY to erase the display from row 2,
C column 6, through row 4, column 28.
C-
        ISTATUS = SMG$ERASE_DISPLAY ( DISPLAY1, 2, 6, 4, 28)
        END
```

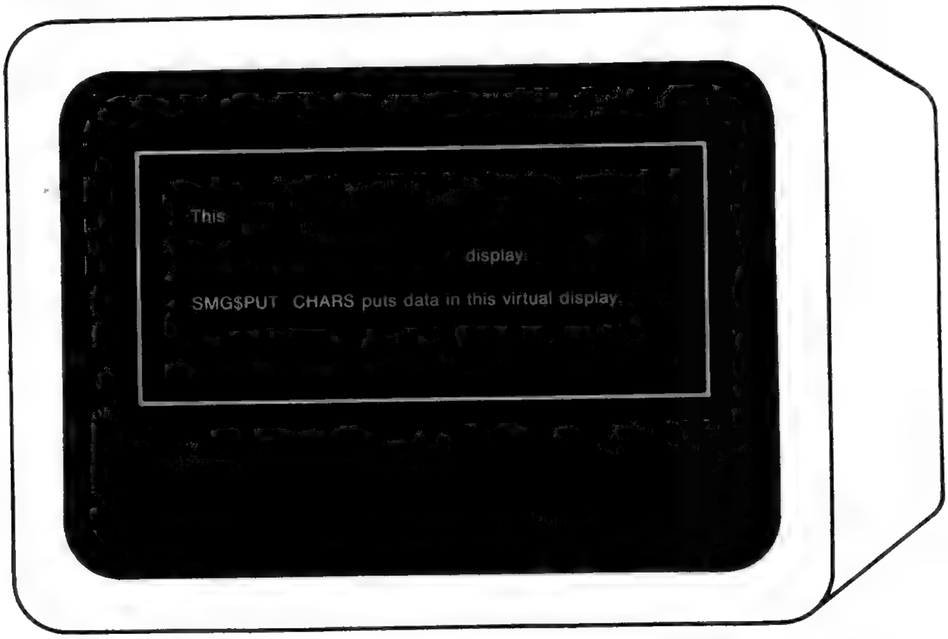
The initial display output by this FORTRAN program is shown in Figure RTL–19.

Figure RTL–19 Initial Output of FORTRAN Program Calling SMG$ERASE_DISPLAY



This output displayed after the call to SMG$ERASE_DISPLAY is shown in Figure RTL-20.

Figure RTL-20 Output Displayed After the Call to SMG$ERASE_DISPLAY



ZK-4115-85

SMG$ERASE_LINE—Erase Line

SMG$ERASE_LINE erases all or part of a line in a virtual display.

FORMAT

SMG$ERASE_LINE *display-id [,line-number] [,column-number]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the virtual display to be affected. The **display-id** argument is the address of an unsigned longword that contains the display identifier. **Display-id** is returned by SMG$CREATE_VIRTUAL_DISPLAY.

line-number

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the line at which the erase operation starts. The **line-number** argument is the address of a signed longword integer that contains the number of the row at which the erasure starts. If omitted, **column-number** is also ignored and the current cursor position is used.

column-number

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the column at which the erase operation starts. The **column-number** argument is the address of a signed longword integer that contains the number of the column at which the erasure starts. If omitted, **row-number** is also ignored and the current cursor position is used.

DESCRIPTION

SMG$ERASE_LINE erases a line from the specified starting position to the end of the line. If you do not specify a starting position, SMG$ERASE_LINE erases text from the current virtual cursor position to the end of the line. The virtual cursor remains at the first blank position after the erased text.

CONDITION VALUES RETURNED

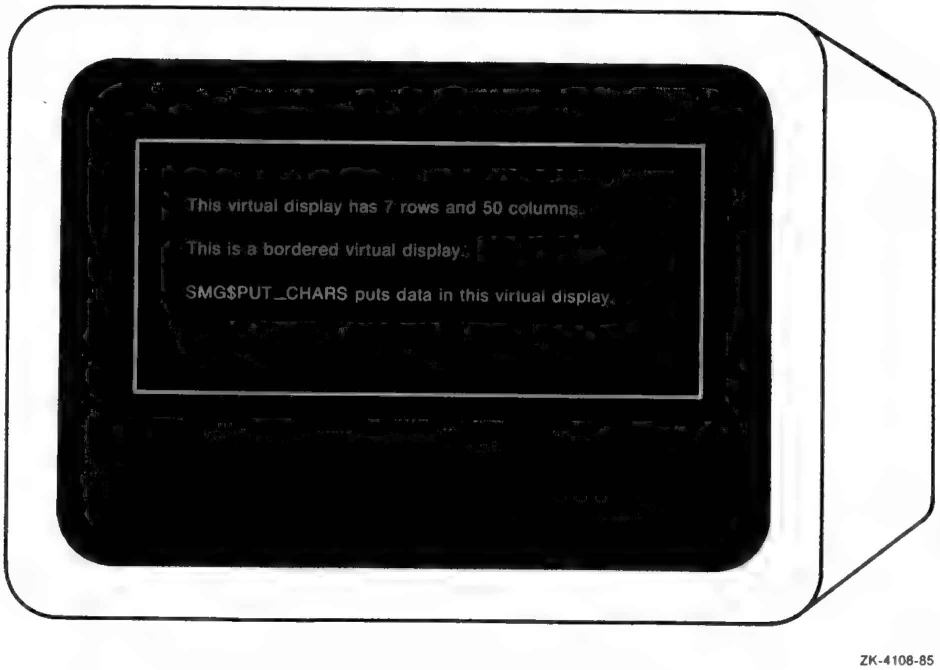
SS$_NORMAL	Normal successful completion.
SMG$_INVDIS_ID	Invalid **display-id**.
SMG$_INVCOL	Invalid column number. The specified column is outside the virtual display.
SMG$_INVROW	Invalid row number. The specified row is outside the virtual display.
SMG$_WRONUMARG	Wrong number of arguments.

EXAMPLE

```
C+
C This FORTRAN example program demonstrates the use of
C SMG$ERASE_LINE.
C-
        INCLUDE '($SMGDEF)'
        INTEGER SMG$CREATE_VIRTUAL_DISPLAY, SMG$CREATE_PASTEBOARD
        INTEGER SMG$PASTE_VIRTUAL_DISPLAY,  SMG$PUT_CHARS,
     1          SMG$ERASE_DISPLAY
        INTEGER DISPLAY1, PASTE1, ROWS, COLUMNS

C+
C Use SMG$CREATE_VIRTUAL_DISPLAY to create a virtual display
C with a border.
C-
        ROWS = 7
        COLUMNS = 50
        ISTATUS = SMG$CREATE_VIRTUAL_DISPLAY
     1          (ROWS, COLUMNS, DISPLAY1, SMG$M_BORDER)
        IF (.NOT. ISTATUS) WRITE (6, 900) 'SMG$CREATE_VIRTUAL_DISPLAY',
     1          ISTATUS

C+
C Call SMG$CREATE_PASTEBOARD to create the pasteboard.
C-
        ISTATUS = SMG$CREATE_PASTEBOARD (PASTE1)
        IF (.NOT. ISTATUS) WRITE (6, 900) 'SMG$CREATE_PASTEBOARD',
     1          ISTATUS

C+
C Put data in the virtual display by calling SMG$PUT_CHARS.
C-

        ISTATUS = SMG$PUT_CHARS ( DISPLAY1,
     1          ' This virtual display has 7 rows and 50 columns.', 2, 1)
        ISTATUS = SMG$PUT_CHARS ( DISPLAY1,
     1          ' This is a bordered virtual display.', 4, 1)
        ISTATUS = SMG$PUT_CHARS ( DISPLAY1,
     1          ' SMG$PUT_CHARS puts data in this virtual display.', 6,
     1          1)

C+
C Use SMG$PASTE_VIRTUAL_DISPLAY to paste the virtual display.
C-
        ISTATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DISPLAY1, PASTE1, 4, 15)

900     FORMAT (' Routine ', A, ' returned a status of ', Z8)

C+
C Call SMG$ERASE_LINE to erase line 2, and then again to
C erase the last 4 words on line 4.
C-
        ISTATUS = SMG$ERASE_LINE ( DISPLAY1, 2, 1)
        ISTATUS = SMG$ERASE_LINE ( DISPLAY1, 4, 9)
        END
```

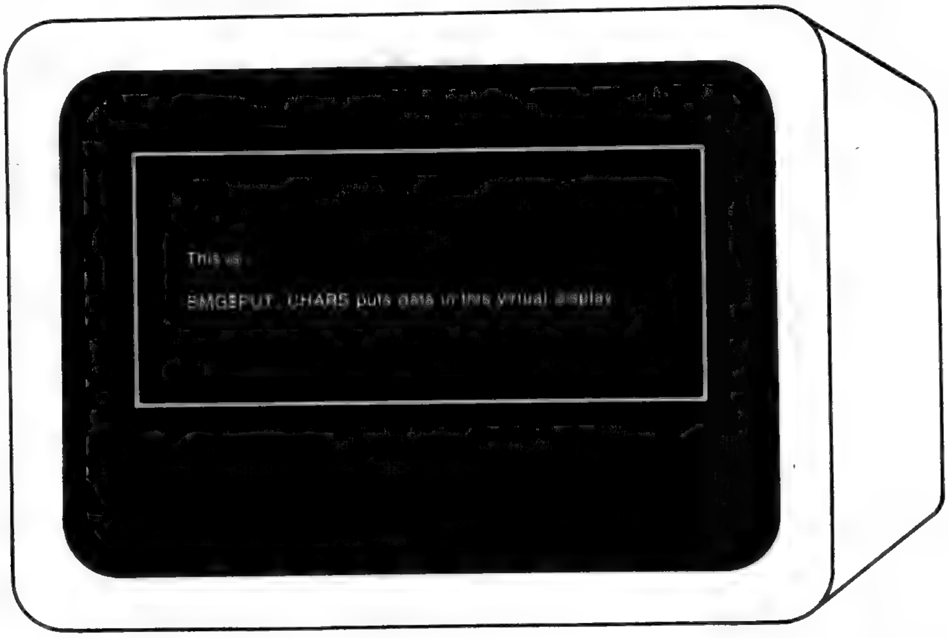
The initial output generated by the FORTRAN program is shown in Figure RTL–21.

Figure RTL–21 Initial Output Generated by FORTRAN Program Calling SMG$ERASE_LINE



The output generated after the call to SMG$ERASE_LINE is shown in Figure RTL–22.

Figure RTL–22 Output Generated After the Call to SMG$ERASE_LINE



SMG$ERASE_PASTEBOARD—Erase Pasteboard

SMG$ERASE_PASTEBOARD erases a pasteboard; that is, it clears the screen.

FORMAT

SMG$ERASE_PASTEBOARD *pasteboard-id*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENT

pasteboard-id
VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the pasteboard to be erased. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier. **Pasteboard-id** is returned by SMG$CREATE_PASTEBOARD.

DESCRIPTION

SMG$ERASE_PASTEBOARD erases the specified pasteboard. The physical cursor is left at position 1,1. If there are any virtual displays pasted to the pasteboard, they will be redrawn the next time the Screen Management Facility is used to output to the pasteboard.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_INVPAS_ID	Invalid **pasteboard-id**.

EXAMPLE

```
C+
C This FORTRAN example program demonstrates how to use
C SMG$CREATE_VIRTUAL_DISPLAY.
C-
        IMPLICIT INTEGER*4 (A-Z)
        CHARACTER*80    OUT_STR,TRIM_STR
        CHARACTER*18    PROMPT          /'Please enter data '/
        SMG$M_BOLD = 1
        SMG$M_REVERSE = 2
        SMG$M_BLINK = 4
        SMG$M_UNDERLINE = 8
C+
C Establish the terminal keyboard as the virtual keyboard
C by calling SMG$CREATE_VIRTUAL_KEYBOARD.
```

```
C-
        STATUS = SMG$CREATE_VIRTUAL_KEYBOARD(KEYBOARD_ID,,)
        IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Establish the terminal screen as a pasteboard using
C SMG$CREATE_PASTEBOARD.
C-
        STATUS = SMG$CREATE_PASTEBOARD (NEW_PID,,,)
        IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Establish a virtual display region by
C calling SMG$CREATE_VIRTUAL_DISPLAY.
C-
        STATUS = SMG$CREATE_VIRTUAL_DISPLAY (5,80,DISPLAY_ID,,)
        IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Paste the virtual display to the screen, starting at
C row 10, column 15.  To paste the virtual display, use
C SMG$PASTE_VIRTUAL_DISPLAY.
C-
        STATUS = SMG$PASTE_VIRTUAL_DISPLAY(DISPLAY_ID,NEW_PID,10,15)
        IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Prompt the user for input, and accept that input using
C SMG$READ_STRING.
C-
        STATUS = SMG$READ_STRING(KEYBOARD_ID,OUT_STR,PROMPT,,,,,,)
        IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Clear the screen using SMG$ERASE_PASTEBOARD.
C-
        STATUS = SMG$ERASE_PASTEBOARD (NEW_PID)
        IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Trim any trailing blanks from the user input
C by calling STR$TRIM.
C-
        STATUS = STR$TRIM(TRIM_STR,OUT_STR,STR_LEN)
        IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Display the data input by the user using SMG$PUT_CHARS
C and SMG$PUT_LINE.
C-
        STATUS = SMG$PUT_CHARS(DISPLAY_ID,'You entered: ',,,,,)
        IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
        STATUS = SMG$PUT_LINE(DISPLAY_ID,TRIM_STR(1:STR_LEN),,
        1                     SMG$M_REVERSE,0,,)
        IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
        END
```

This FORTRAN program calls Run-Time Library Screen Management routines to format screen output, and to accept and display user input.

SMG$FIND_CURSOR_DISPLAY Find Display That Contains The Cursor

SMG$FIND_CURSOR_DISPLAY returns the identifier of the most recently pasted virtual display that contains the physical cursor.

FORMAT

SMG$FIND_CURSOR_DISPLAY *pasteboard-id ,returned-display-id*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

pasteboard-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the pasteboard (physical screen) in which the cursor is to be found. The **pasteboard-id** argument is the address of a longword that contains the pasteboard identifier. **Pasteboard-id** is returned by SMG$CREATE_PASTEBOARD.

returned-display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Receives the identifier of the display in which the cursor was found. The **returned-display-id** argument is the address of a longword into which is written the display identifier.

DESCRIPTION

SMG$FIND_CURSOR_DISPLAY determines which virtual display contains the physical cursor on a specified pasteboard, and returns the virtual display's identifier. SMG$FIND_CURSOR_DISPLAY returns the display-id of the most recently pasted virtual display which contains the physical cursor. If no virtual display contains the physical cursor, this routine returns a 0, which is an invalid display identifier.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVPAS_ID	Invalid **pasteboard-id**.
SMG$_WRONUMARG	Wrong number of arguments.

SMG$FLUSH_BUFFER—Flush Buffer

SMG$FLUSH_BUFFER flushes all buffered output to the terminal.

FORMAT

SMG$FLUSH_BUFFER *pasteboard-id*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENT

pasteboard-id

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Specifies the pasteboard to be flushed. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier. **Pasteboard-id** is returned by SMG$CREATE_PASTEBOARD.

DESCRIPTION

SMG$FLUSH_BUFFER causes all buffered output to be sent to the terminal immediately. The calling program would normally call this routine just before performing some cpu-intensive calculations, or whenever the terminal screen must be up-to-date.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_INVPAS_ID	Invalid **pasteboard-id**.

SMG$GET_BROADCAST_MESSAGE Get Broadcast Message

SMG$GET_BROADCAST_MESSAGE determines whether a message has been broadcast to the pasteboard and returns the message.

FORMAT

SMG$GET_BROADCAST_MESSAGE *pasteboard-id [,message] [,message-length]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

pasteboard-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the terminal to be checked for the presence of a broadcast message. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier. **Pasteboard-id** is returned by SMG$CREATE_PASTEBOARD.

message

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

A string that receives the broadcast message, if such a message is available. The **message** argument is the address of a descriptor that points to the storage into which the message text is written. If this argument is omitted, the broadcast message is discarded.

message-length

VMS Usage: **word_signed**
type: **word integer (signed)**
access: **write only**
mechanism: **by reference**

Receives the actual length of the broadcast message. The **message-length** argument is the address of a word into which is written the length of the message.

DESCRIPTION

SMG$GET_BROADCAST_MESSAGE determines whether any broadcast messages have been sent to the specified terminal while broadcast trapping was enabled, and if so, returns the message in the **message** argument. If no broadcast messages have been sent to the terminal, SMG$GET_BROADCAST_MESSAGE returns the success status SMG$_NO_MORMSG.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_NO_MORMSG	Successful completion. No more messages to be returned.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_INVPAS_ID	Invalid **pasteboard-id**.

Any condition values returned by LIB$SCOPY_DXDX.

SMG$GET_CHAR_AT_PHYSICAL_CURSOR Return Character At Cursor

SMG$GET_CHAR_AT_PHYSICAL_CURSOR returns the character at the current physical cursor position.

FORMAT

SMG$GET_CHAR_AT_PHYSICAL_CURSOR *pasteboard-id ,character*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

pasteboard-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the pasteboard on which to retrieve the character. The **pasteboard-id** argument is the address of a longword that contains the pasteboard identifier.

Pasteboard-id is returned by SMG$CREATE_PASTEBOARD.

character

VMS Usage: **byte_unsigned**
type: **byte (unsigned)**
access: **write only**
mechanism: **by reference**

Returned character code. The **character** argument is the address of an unsigned byte into which is written the character's ASCII code.

DESCRIPTION

SMG$GET_CHAR_AT_PHYSICAL_CURSOR returns the character that occupies the screen position corresponding to the current physical cursor position.

Note: **If the Screen Management Facility has not written to the screen location occupied by the physical cursor, then the contents of that position are unknown. In this case, the hexadecimal value FF is returned.**

If the returned character has an ASCII value less than 20 (hexadecimal), then it is not a printable character. Rather, it is an internal terminal-independent code denoting what should be displayed at that position (for example, an element of the line-drawing character set). Do not attempt to use this code for subsequent output operations.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_INVPAS_ID	Invalid **pasteboard-id**.

SMG$GET_DISPLAY_ATTR—Get Display Attributes

SMG$GET_DISPLAY_ATTR returns attributes associated with a virtual display.

FORMAT

SMG$GET_DISPLAY_ATTR *display-id [,height] [,width] [,display-attributes] [,video-attributes] [,char-set]*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

display-id

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Specifies the virtual display for which information is requested. The **display-id** argument is the address of an unsigned longword that contains the display identifier. **Display-id** is returned by SMG$CREATE_VIRTUAL_DISPLAY.

height

VMS Usage:	**longword_signed**
type:	**longword integer (signed)**
access:	**write only**
mechanism:	**by reference**

Receives the number of rows in the display. The **height** argument is the address of a signed longword integer into which the height is written.

width

VMS Usage:	**longword_signed**
type:	**longword integer (signed)**
access:	**write only**
mechanism:	**by reference**

Receives the number of columns in the display. The **width** argument is the address of a signed longword integer into which is written the number of columns in the display.

display-attributes

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Receives the current default display attributes (for example, SMG$M_BORDER). The **display-attributes** argument is the address of an unsigned longword into which the current display attributes are written.

video-attributes

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Receives the current default video attributes. The **video-attributes** argument is the address of an unsigned longword into which the current video attributes are written.

Valid video attributes are as follows:

SMG$M_BLINK	Displays blinking characters
SMG$M_BOLD	Displays characters in higher-than-normal intensity
SMG$M_REVERSE	Displays characters in reverse video, that is, using the opposite default rendition of the virtual display
SMG$M_UNDERLINE	Displays underlined characters

char-set

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the default character set for all text in this virtual display. The **char-set** argument is the address of an unsigned longword that specifies the character set. At this time, the only valid value is SMG$C_ASCII, which is also the default.

DESCRIPTION

SMG$GET_DISPLAY_ATTR returns the attributes of a virtual display.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVDIS_ID	Invalid **display-id**.
SMG$_WRONUMARG	Wrong number of arguments.

SMG$GET_KEY_DEF—Get Key Definition

SMG$GET_KEY_DEF returns the key definition for a specified key.

FORMAT

SMG$GET_KEY_DEF *key-table-id ,key-name [,if-state] [,attributes] [,equiv-string] [,state-string]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

key-table-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the key table from which you are extracting a definition. The **key-table-id** argument is the address of an unsigned longword that contains the key table identifier. **Key-table-id** is returned by SMG$CREATE_KEY_TABLE.

key-name

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Specifies the name of the key associated with the definition. The **key-name** argument is the address of a descriptor pointing to the key name.

Table 3-1 lists the valid key names.

if-state

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Specifies the current state name in effect after the key is pressed. The **if-state** argument is the address of a descriptor pointing to the state name.

See SMG$ADD_KEY_DEF for more information.

attributes

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **write only**

mechanism: **by reference**
Receives the attributes bit mask for this key definition. The **attributes** argument is the address of a longword into which is written the bit mask describing the key's attributes.

Valid values are as follows:

SMG$M_KEY_NOECHO	If set, this bit specifies that **equiv_string** is not to be echoed when this key is pressed. If clear, **equiv_string** is echoed. If SMG$M_KEY_TERMINATE is not set, SMG$M_KEY_NOECHO is ignored.
SMG$M_KEY_TERMINATE	If set, this bit specifies that when this key is pressed (as qualified by **if-state**), the input line is complete and more characters should not be accepted. If clear, more characters may be accepted.
SMG$M_KEY_LOCKSTATE	If set, and if **state-string** is specified, the state name specified by **state-string** remains the current state until explicitly changed by a subsequent keystroke whose definition includes a **state-string**. If clear, the state name specified by **state-string** remains in effect only for the next defined key stroke.
SMG$M_KEY_PROTECTED	If set, this bit specifies that this key definition cannot be modified or deleted. If clear, the key definition can be modified or deleted.

equiv-string

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**
Receives the equivalence string for this key definition. The **equiv-string** argument is the address of a descriptor pointing to the storage into which is written the equivalence string.

state-string

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**
Receives the new state name, if any, which is set by this key definition. The **state-string** argument is the address of a descriptor pointing to the storage into which is written the new state string.

DESCRIPTION

SMG$GET_KEY_DEF returns the key definition associated with a specified **key-name** and **if-state**.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVKTB_ID	Invalid **key-table-id**.
SMG$_KEYNOTDEF	Key not defined.
SMG$_WRONUMARG	Wrong number of arguments.

Any condition values returned by LIB$SCOPY_DXDX.

SMG$GET_KEYBOARD_ATTRIBUTES Get Keyboard Attributes

SMG$GET_KEYBOARD_ATTRIBUTES gets information about a virtual keyboard and leaves it in a user-supplied area: the keyboard information table (KIT).

FORMAT

SMG$GET_KEYBOARD_ATTRIBUTES *keyboard-id ,p-kit ,p-kit-size*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

keyboard-id

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Keyboard identifier. The **keyboard-id** argument is the address of an unsigned longword containing the identification of the virtual keyboard from which to read. A virtual-keyboard is created by calling the SMG$CREATE_VIRTUAL_KEYBOARD routine.

p-kit

VMS Usage:	**address**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by reference**

Address of the keyboard information table. The **p-kit** argument is the address of an unsigned longword that contains a pointer to the keyboard information table (KIT). The KIT is a byte block whose size and field references are described in $SMGDEF. It is the caller's responsibility to allocate the correct size block and to pass its address to this routine.

The values in the **p-kit** can be accessed through the following symbolic

names:

SMG$L_DEVCHAR	Device characteristics (longword)
SMG$L_DEVDEPEND	Specific characteristics 1 (longword)
SMG$L_DEVDEPEND2	Specific characteristics 2 (longword)
SMG$B_DEVCLASS	Device class (byte) — for example, DC$_TERM
SMG$B_RECALL_SIZE	Size of recall buffer (byte) (*)
SMG$B_PHY_DEVTYPE	Physical device type (byte) — for example, DT$_VT100
SMG$B_TYPEAHEAD_CHAR	First character in type-ahead buffer (byte) (*)
SMG$W_WIDTH	Terminal width (word)
SMG$W_TYPEAHEAD_COUNT	Number of characters in type-ahead buffer (word) (*)

Items marked with an asterisk (*) will be zero unless the device is a terminal (DEVCLASS = DC$_TERM).

p-kit-size

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Size of the keyboard information table. The **p-kit-size** argument is the address of an unsigned longword containing the size of the KIT in bytes.

The size you specify must be exact. You can specify this size with the symbolic constant SMG$S_KEYBOARD_INFO_BLOCK.

DESCRIPTION

SMG$GET_KEYBOARD_ATTRIBUTES retrieves information about a virtual keyboard and leaves this information in the keyboard information table (KIT). The KIT is a user-supplied area consisting of a byte block.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVARG	KIT is the wrong size.
SMG$_INVKBD_ID	Invalid **keyboard-id**.

SMG$GET_NUMERIC_DATA
Get Numeric Terminal Data

SMG$GET_NUMERIC_DATA accesses TERMTABLE.EXE and returns the numeric sequence that causes a terminal to perform a specified operation.

FORMAT

SMG$GET_NUMERIC_DATA *termtable-address ,request-code ,buffer-address*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

termtable-address

VMS Usage: **address**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the address of the TERMTABLE entry for the desired terminal. The **termtable-address** argument is the address of an unsigned longword that contains the address of TERMTABLE information.

Before calling SMG$GET_NUMERIC_DATA, you must obtain this terminal table address by calling either SMG$INIT_TERM_TABLE or SMG$INIT_TERM_TABLE_BY_TYPE.

request-code

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Request code which specifies the desired capability. The **request-code** argument is a signed longword constant containing this request code. The request code is of the form SMG$K_code where code corresponds to a keyword in the terminal capabilities table, for example, ANSI_CRT.

See Tables 3–2, 3–3, and 3–4 in Part I of this manual for valid capability fields.

buffer-address

VMS Usage: **address**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Address of the first byte of the longword to which SMG$GET_NUMERIC_DATA writes the numeric capability data. The **buffer-address** argument is an unsigned longword which contains the address of this buffer.

DESCRIPTION

SMG$GET_NUMERIC_DATA extracts the requested numeric information from a specified terminal table. Before calling SMG$GET_NUMERIC_DATA, you must obtain that terminal table address by calling either SMG$INIT_TERM_TABLE or SMG$INIT_TERM_TABLE_BY_TYPE.

CONDITION VALUES RETURNED

SMG$_INVTERTAB	Invalid terminal table address.
SMG$_INVREQCOD	Invalid request code.
SS$_NORMAL	Normal successful completion.

SMG$GET_PASTEBOARD_ATTRIBUTES Get Pasteboard Attributes

SMG$GET_PASTEBOARD_ATTRIBUTES gets pasteboard attributes and stores them in the pasteboard information table.

FORMAT

SMG$GET_PASTEBOARD_ATTRIBUTES *pasteboard-id ,pb-info-table ,pb-info-table-size*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

pasteboard-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the pasteboard for which information is requested. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

Pasteboard-id is returned by SMG$CREATE_VIRTUAL_PASTEBOARD.

pb-info-table

VMS Usage: **vector_byte_unsigned**
type: **byte (unsigned)**
access: **write only**
mechanism: **by reference, array reference**

Receives the pasteboard attributes. The **pb-info-table** argument is the address of an array of unsigned bytes into which is written the pasteboard attributes.

The values in the **pb-info-table** can be accessed through the following symbolic names:

SMG$L_DEVCHAR	Device characteristics (longword)
SMG$L_DEVDEPEND	Specific characteristics 1 (longword)
SMG$L_DEVDEPEND2	Specific characteristics 2 (longword)
SMG$B_DEVCLASS	Device class (byte)— for example, DC$_TERM

SMG$B_SMG_DEVTYPE	Internal SMG device type (byte). The four possible values for SMG$B_SMG_DEVTYPE are: SMG$K_UNKNOWN SMG$K_VTFOREIGN SMG$K_HARDCOPY SMG$K_VTTERMTABLE
SMG$B_PHY_DEVTYPE	Physical device type (byte)—for example, DT$_VT100. The possible values for SMG$B_PHY_DEVTYPE are defined in $TTDEF in STARLET.
SMG$B_ROWS	Number of rows on device (byte)
SMG$W_WIDTH	Terminal width (word)
SMG$B_COLOR	Color setting (byte). The three possible values for SMG$B_COLOR are: SMG$C_WHITE SMG$C_BLACK SMG$C_UNKNOWN
SMG$B_PARITY	Parity attributes (byte)—this field is zero if the device is not a terminal
SMG$W_SPEED	Terminal speed (word)—this field is zero if the device is not a terminal
SMG$W_FILL	Fill characteristics (word)—this field is zero if the device is not a video terminal
SMG$W_CURSOR_ROW	Row containing physical cursor (word)
SMG$W_CURSOR_COL	Column containing physical cursor (word)
SMG$L_CURSOR_DID	Display id of topmost display containing physical cursor (longword)

pb-info-table-size

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the number of bytes in the **pb-info-table**. The **pb-info-table-size** argument is the address of an unsigned longword that contains the size (in bytes) of the **pb-info-table**.

The size you specify must be exact. You can specify this size with the symbolic constant SMG$S_PASTEBOARD_INFO_BLOCK.

DESCRIPTION

SMG$GET_PASTEBOARD_ATTRIBUTES gets pasteboard attributes and stores them in the pasteboard information table.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVARG	Incorrect size specified in **pb-info-table-size**.
SMG$_WRONUMARG	Wrong number of arguments.

SMG$GET_PASTING_INFO—Return Pasting Information

Provided that the specified virtual display is currently pasted, the row and column of the pasting are returned.

FORMAT

SMG$GET_PASTING_INFO *display-id ,pasteboard-id ,pasted-flag [,pasteboard-row] [,pasteboard-col]*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

display-id

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Identifier of the virtual display to be examined. The **display-id** argument is the address of an unsigned longword containing the identifier of this virtual display.

pasteboard-id

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Identifier of the pasteboard on which the virtual display is pasted. The **pasteboard-id** argument is the address of an unsigned longword containing the identifier of this pasteboard.

pasted-flag

VMS Usage:	**boolean**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by reference**

Flag indicating the status of the specified virtual display with respect to the specified pasteboard. The **pasted-flag** argument is the address of an unsigned longword indicating this status.

If the **pasted-flag** argument is set to 1, the virtual display specified by **display-id** is pasted to the pasteboard specified by the **pasteboard-id** argument. If **pasted-flag** is zero, the virtual display is not pasted to the specified pasteboard.

pasteboard-row

VMS Usage: **longword_signed**
type: **longword (signed)**
access: **write only**
mechanism: **by reference**

Row of the pasteboard that contains row 1 of the specified virtual display. The optional **pasteboard-row** argument is the address of a signed longword containing the row number of the pasteboard row that contains the first row of the virtual display.

pasteboard-col

VMS Usage: **longword_signed**
type: **longword (signed)**
access: **write only**
mechanism: **by reference**

Column of the pasteboard that contains column 1 of the specified virtual display. The optional **pasteboard-col** argument is the address of a signed longword containing the column number of the pasteboard column that contains the first column of the virtual display.

DESCRIPTION

SMG$GET_PASTING_INFO first checks to see if the virtual display specified by **display-id** is pasted with respect to the pasteboard specified by **pasteboard-id**. If this virtual display is pasted with respect to the pasteboard, SMG$GET_PASTING_INFO returns the row and column numbers of the pasteboard that correspond to row and column 1 of the pasted virtual display.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVDIS_ID	Invalid **display-id**.
SMG$_INVPAS_ID	Invalid **pasteboard-id**.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_ILLBATFNC	Display is batched.

SMG$GET_TERM_DATA—Get Terminal Data

SMG$GET_TERM_DATA accesses TERMTABLE.EXE and returns the character sequence that causes a terminal to perform a specified operation.

FORMAT

SMG$GET_TERM_DATA *termtable-address ,request-code ,max-buffer-length ,return-length ,buffer-address [,input-argument-vector]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

termtable-address

VMS Usage: **address**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the address of the TERMTABLE entry for the desired terminal. The **termtable-address** argument is the address of an unsigned longword that contains the address of TERMTABLE information.

Termtable-address is returned by SMG$INIT_TERM_TABLE or SMG$INIT_TERM_TABLE_BY_TYPE.

request-code

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Longword constant of the form SMG$K_code, where code is the name of the desired capability field. The **request-code** argument is the address of an unsigned longword that contains the request code.

See Tables 3-2, 3-3, and 3-4, in Part I of this manual for valid capability fields.

max-buffer-length

VMS Usage:	**longword_signed**
type:	**longword integer (signed)**
access:	**read only**
mechanism:	**by reference**

Maximum length of the buffer into which the requested capability data is written. The **max-buffer-length** argument is the address of a signed longword integer that contains the maximum number of bytes that can be written into the buffer.

return-length

VMS Usage:	**longword_signed**
type:	**longword integer (signed)**
access:	**write only**
mechanism:	**by reference**

Receives the number of bytes actually written into the buffer. The **return-length** argument is the address of a signed longword integer into which is written the number of bytes transferred into the buffer.

buffer-address

VMS Usage:	**address**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by reference**

Address of the first byte of the buffer which is to receive the capability data. The **buffer-address** argument is an unsigned longword that contains the address of the buffer.

input-argument-vector

VMS Usage:	**address**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference, array reference**

Address of a list of longwords used for capabilities that require a variable number of arguments, and for those that require substitution or arithmetic operations on an argument. The **input-argument-vector** argument is the address of an array of unsigned longwords that contains capability arguments. The first longword must contain the number of arguments that follow.

DESCRIPTION

SMG$GET_TERM_DATA should be used only when you perform direct (non-SMG$) I/O to terminals. It accesses the TERMTABLE.EXE entry for the specified type of terminal and returns the character sequence that performs the specified operation. It is up to you to send this character sequence to the terminal.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVTERTAB	Invalid terminal table address.
SMG$_INVREQCOD	Invalid request code.

SMG$HOME_CURSOR—Home Cursor

SMG$HOME_CURSOR moves the virtual cursor to the specified corner of a virtual display.

FORMAT

SMG$HOME_CURSOR *display-id [,position]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the virtual display in which the virtual cursor is moved. The **display-id** argument is the address of a longword that contains the display identifier. **Display-id** is returned by SMG$CREATE_VIRTUAL_DISPLAY.

position

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the point to which the virtual cursor moves. The **position** argument is the address of a longword that contains the position code.

Valid codes for **position** are as follows:

Code	Meaning
SMG$C_UPPER_LEFT	Row 1 column 1 (the upper left-hand corner). This is the default if position is not specified.
SMG$C_LOWER_LEFT	Row n column 1 (where n is the number of rows in the display. That is, the lower left-hand corner. It is useful to specify this position when accepting input for an upward-scrolling virtual display).
SMG$C_UPPER_RIGHT	Row 1 column m (where m is the number of columns in the display. That is, the upper right-hand corner).
SMG$C_LOWER_RIGHT	Row n column m (where n is the number of rows and m is the number of columns in the display. That is, the lower right-hand corner).

DESCRIPTION

SMG$HOME_CURSOR moves the virtual cursor to a corner of the specified virtual display, according to the code specified in the **position** argument. The caller need not know the dimensions of the virtual display.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVDIS_ID	Invalid **display-id**.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_INVARG	Invalid argument.

SMG$INIT_TERM_TABLE—Initialize Terminal Table

SMG$INIT_TERM_TABLE initializes the TERMTABLE database for the terminal named, so that subsequent calls to SMG$GET_TERM_DATA can extract information and command strings for that terminal.

FORMAT

SMG$INIT_TERM_TABLE *terminal-name, term-entry-address*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

terminal-name

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Specifies the name of the terminal. The **terminal-name** argument is the address of a descriptor pointing to the terminal name. The name must be an entry in TERMTABLE.EXE

term-entry-address

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Address of the entry for a particular type of terminal in TERMTABLE.EXE. The **term-entry-address** argument is the address of an unsigned longword that contains the address of TERMTABLE information.

You use this address when calling the SMG$GET_TERM_DATA procedure for that type of terminal. **Term-entry-address** is returned by SMG$INIT_TERM_TABLE or SMG$INIT_TERM_TABLE_BY_TYPE.

DESCRIPTION

SMG$INIT_TERM_TABLE initializes the TERMTABLE database for the terminal named, so that subsequent calls to SMG$GET_TERM_DATA can extract information and command strings for that terminal. This routine should be used only when you perform direct (non-SMG$) I/O to terminals.

SMG$INIT_TERM_TABLE first searches for TERMTABLE in the area logically named TERM$TABLOC. If TERMTABLE is not found there, the routine searches the global section SMG$TERMTABLE.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_PRISECMAP	Successful completion. The definition was found in a private TERMTABLE.
SMG$_GBLSECMAP	Successful completion. The definition was found in the global TERMTABLE.
SMG$_UNDTERNOP	Undefined terminal. No definition was found for the terminal and no private TERMTABLE was found.
SMG$_UNDTERNOS	Undefined terminal. No definition was found for the terminal and no system TERMTABLE was found.
SMG$_UNDTERNAM	Undefined terminal name.

SMG$INIT_TERM_TABLE_BY_TYPE Initialize TERMTABLE By VMS Terminal Type

SMG$INIT_TERM_TABLE_BY_TYPE initializes the TERMTABLE database for the terminal named, so that subsequent calls to SMG$GET_TERM_DATA can extract information and command strings for that terminal.

FORMAT

SMG$INIT_TERM_TABLE_BY_TYPE *terminal-type ,term-entry-address [,terminal-name]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

terminal-type

VMS Usage: **byte_signed**
type: **byte integer (signed)**
access: **read only**
mechanism: **by reference**

The device type of the terminal, as designated by a VMS symbolic terminal type or by another value returned by the $GETDVI system service. The **terminal-type** argument is the address of a signed byte integer that contains the terminal type.

term-entry-address

VMS Usage: **address**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Address of the entry for a particular type of terminal in TERMTABLE.EXE. The **term-entry-address** argument is the address of an unsigned longword into which is written the address of a terminal entry.

You use this address when calling the SMG$GET_TERM_DATA procedure for that type of terminal.

terminal-name

VMS Usage: **device_name**
type: **character string**
access: **write only**
mechanism: **by descriptor**

A string into which is written the terminal name associated with the device type. The **terminal-name** argument is the address of a descriptor pointing to the storage into which the terminal name is written.

DESCRIPTION

SMG$INIT_TERM_TABLE_BY_TYPE initializes the TERMTABLE database for the terminal type specified, so that subsequent calls to SMG$GET_TERM_DATA can extract information and command strings for that type of terminal. This routine should be used only when you perform direct (non-SMG$) I/O to terminals.

SMG$INIT_TERM_TABLE_BY_TYPE first searches for TERMTABLE in the area logically named TERM$TABLOC. If TERMTABLE is not found there, the routine searches the global section SMG$TERMTABLE.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_PRISECMAP	Successful completion. The definition was found in a private TERMTABLE.
SMG$_GBLSECMAP	Successful completion. The definition was found in the global TERMTABLE.
SMG$_UNDTERNOP	Undefined terminal. No definition was found for the terminal and no private TERMTABLE was found.
SMG$_UNDTERNOS	Undefined terminal. No definition was found for the terminal and no system TERMTABLE was found.
SMG$_UNDTERNAM	Undefined terminal name.

SMG$INSERT_CHARS—Insert Characters

SMG$INSERT_CHARS inserts characters into a virtual display.

FORMAT

SMG$INSERT_CHARS *display-id ,string ,row ,column [,rendition-set] [,rendition-complement] [,char-set]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the virtual display affected. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier. **Display-id** is returned by SMG$CREATE_VIRTUAL_DISPLAY.

string

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

The character string to be inserted. The **string** argument is the address of a descriptor that points to the string to be inserted.

row

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

The row position at which to begin the insertion. The **row** argument is the address of a signed longword integer that contains the row number.

column

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

The column position at which to begin the insertion. The **column** argument is the address of a signed longword integer that contains the column number.

rendition-set

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

A mask which denotes video attributes for the characters inserted. The **rendition-set** argument is the address of an unsigned longword that contains a video attributes mask. Each bit attribute in this argument causes the corresponding attribute to be set in the display.

Video attributes which can be manipulated in this manner are as follows:

SMG$M_BLINK	Displays blinking characters
SMG$M_BOLD	Displays characters in higher-than-normal intensity
SMG$M_REVERSE	Displays characters in reverse video, that is, using the opposite default rendition of the virtual display
SMG$M_UNDERLINE	Displays underlined characters

If the same bit is set in both the **rendition-set** and **rendition-complement** arguments, the Screen Management Facility applies the **rendition-set** attribute followed by the **rendition-complement** attribute. Using these two arguments, the caller can exercise independent control over each attribute in a single call.

rendition-complement

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

A mask which denotes video attributes for the characters inserted. The **rendition-complement** argument is the address of an unsigned longword that contains a video attributes mask. Each bit attribute in this argument causes the corresponding attribute to be set in the display. Video attributes which can be manipulated in this manner are the same as those for the **rendition-set** argument.

The following table shows the action taken by the Screen Management Facility for various combinations of **rendition-set** and **rendition-complement** attributes.

Set	Complement	Action
0	0	Attribute unchanged
1	0	Attribute on
0	1	Attribute set to complement of default setting
1	1	Attribute off

char-set

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the default character set for all text in this virtual display. The **char-set** argument is the address of an unsigned longword that contains

the character set code. At this time, the only valid value is SMG$C_ASCII, which is also the default.

DESCRIPTION

SMG$INSERT_CHARS inserts the specified character string at the **row** and **column** positions specified. Characters to the right of the insertion are shifted to the right. Note that any characters which do not fit on the current line are discarded. The virtual cursor remains at the character position following the last character inserted.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVROW	Invalid row.
SMG$_INVCOL	Invalid column.
SMG$_INVDIS_ID	Invalid **display-id**.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_INVARG	Unrecognized rendition code.
LIB$_INVSTRDES	Invalid string descriptor.

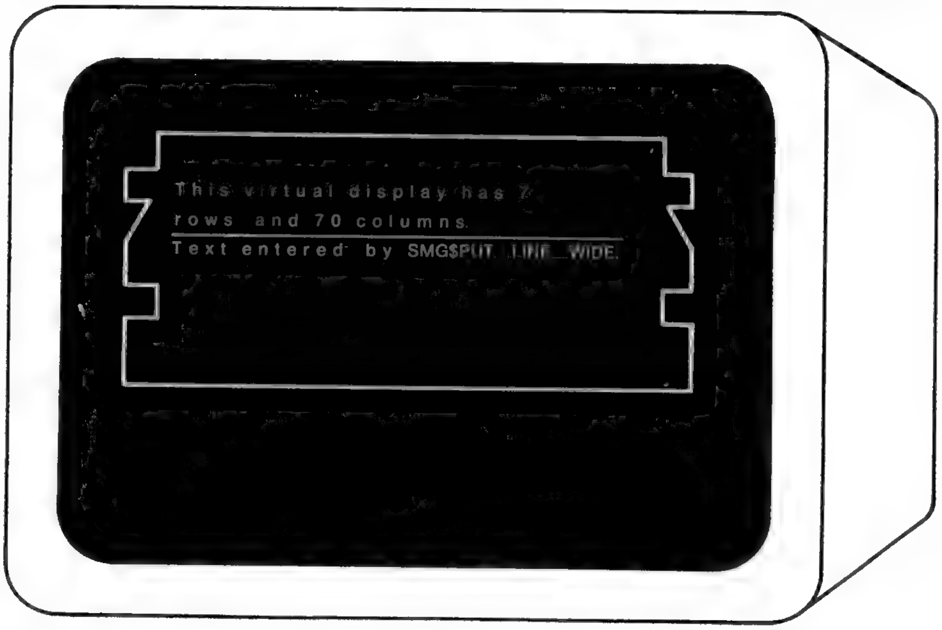
EXAMPLE

```
C+
C This FORTRAN example program illustrates the use of SMG$INSERT_CHARS.
C-
        INCLUDE '($SMGDEF)'
        INTEGER SMG$CREATE_VIRTUAL_DISPLAY, SMG$CREATE_PASTEBOARD
        INTEGER SMG$PASTE_VIRTUAL_DISPLAY,  SMG$PUT_CHARS,
     1          SMG$ERASE_DISPLAY
        INTEGER DISPLAY1, PASTE1, ROWS, COLUMNS

C+
C Use SMG$CREATE_VIRTUAL_DISPLAY to create a virtual display
C with a border.
C-
        ROWS = 7
        COLUMNS = 50
        ISTATUS = SMG$CREATE_VIRTUAL_DISPLAY
     1          (ROWS, COLUMNS, DISPLAY1, SMG$M_BORDER)
        IF (.NOT. ISTATUS) WRITE (6, 900) 'SMG$CREATE_VIRTUAL_DISPLAY',
     1          ISTATUS

C+
C Call SMG$CREATE_PASTEBOARD to create the pasteboard.
C-
        ISTATUS = SMG$CREATE_PASTEBOARD (PASTE1)
        IF (.NOT. ISTATUS) WRITE (6, 900) 'SMG$CREATE_PASTEBOARD',
     1          ISTATUS

C+
C Put data in the virtual display by calling SMG$PUT_CHARS.
C-
        ISTATUS = SMG$PUT_CHARS ( DISPLAY1,
     1          ' This virtual display has 7 rows and 50 columns.', 2, 1)
        ISTATUS = SMG$PUT_CHARS ( DISPLAY1,
     1          ' This is a bordered virtual display.', 4, 1)
        ISTATUS = SMG$PUT_CHARS ( DISPLAY1,
     1          ' SMG$PUT_CHARS puts data in this virtual display.', 6,
     1          1)

C+
C Use SMG$PASTE_VIRTUAL_DISPLAY to paste the virtual display.
C-
```

```
        ISTATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DISPLAY1, PASTE1, 4, 15)
900     FORMAT (' Routine ', A, ' returned a status of ', Z8)
C+
C Call SMG$INSERT_CHARS to add a row 1 of text, starting at column 6.
C Underline these characters.
C-
        ISTATUS = SMG$INSERT_CHARS ( DISPLAY1,
     1          'This is a new row.', 1, 6, SMG$M_UNDERLINE)
C+
C Calling SMG$INSERT_CHARS again, add text to row 6.
C Note that there will be some characters that will no
C longer fit on the line. They will be discarded. The
C new text will be bolded.
C-
        ISTATUS = SMG$INSERT_CHARS ( DISPLAY1,
     1          'to this bordered display.', 6, 28, SMG$M_BOLD)
        END
```

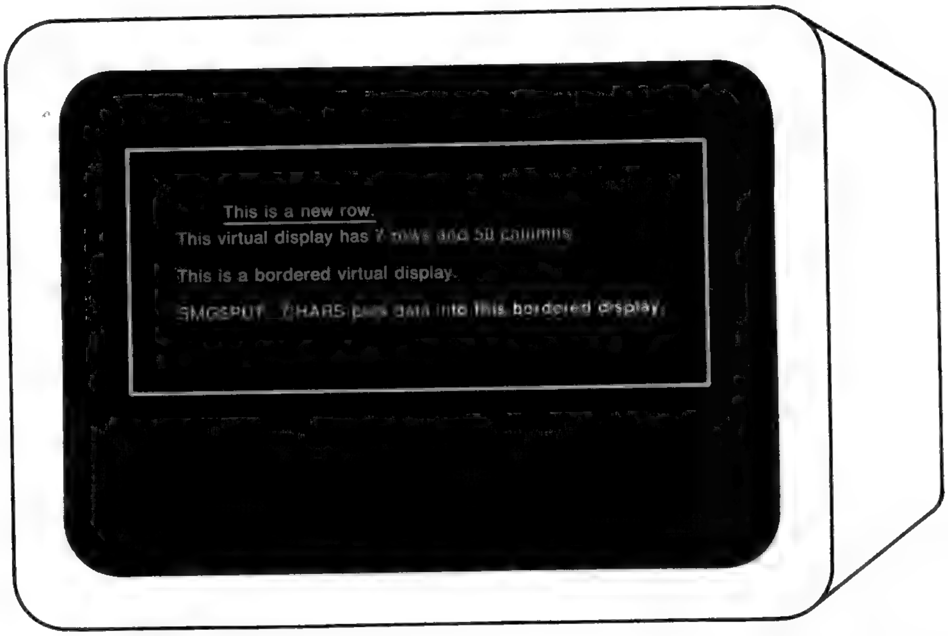
The output generated by this FORTRAN program before the call to SMG$INSERT_CHARS is shown in Figure RTL-23.

Figure RTL-23 Output Generated by FORTRAN Program Before the Call to SMG$INSERT_CHARS



The output generated by this FORTRAN program after the call to SMG$INSERT_CHARS is shown in Figure RTL-24.

Figure RTL–24 Output Generated by FORTRAN Program After the Call to SMG$INSERT_CHARS



SMG$INSERT_LINE—Insert Line

SMG$INSERT_LINE inserts a line into a virtual display and scrolls the display.

FORMAT

SMG$INSERT_LINE *display-id ,line-number [,string] [,direction] [,rendition-set] [,rendition-complement] [,wrap-flag] [,char-set]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the virtual display affected. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

Display-id is returned by SMG$CREATE_VIRTUAL_DISPLAY.

line-number

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the line number at which the string is inserted and at which point scrolling begins. The **line-number** argument is the address of a signed longword integer that contains the line number.

string

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

The character string to be inserted by SMG$INSERT_LINE. The **string** argument is the address of a descriptor pointing to this string.

direction

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the scrolling direction. The **direction** argument is the address of a longword that contains the direction code. Valid values are SMG$M_UP and SMG$M_DOWN. SMG$M_UP is the default.

rendition-set

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Mask that denotes video attributes for the drawn line. The **rendition-set** argument is the address of an unsigned longword that contains a video attributes mask. Each bit attribute in this argument causes the corresponding attribute to be set in the display.

Video attributes that can be manipulated in this manner are as follows:

SMG$M_BLINK	Displays blinking characters
SMG$M_BOLD	Displays characters in higher-than-normal intensity
SMG$M_REVERSE	Displays characters in reverse video, that is, using the opposite default rendition of the virtual display
SMG$M_UNDERLINE	Displays underlined characters

If the same bit is set in both the **rendition-set** and **rendition-complement** arguments, the Screen Management Facility applies the **rendition-set** attribute followed by the **rendition-complement** attribute. Using these two arguments, the caller can exercise independent control over each attribute in a single call.

rendition-complement

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Mask that denotes video attributes for the line drawn. The **rendition-complement** argument is the address of an unsigned longword that contains a video attributes mask.

Each bit attribute in this argument causes the corresponding attribute to be set in the display. Video attributes that can be manipulated in this manner are the same as for the **rendition-set** argument.

The following table shows the action taken by the Screen Management Facility for various combinations of **rendition-set** and **rendition-complement** attributes.

Set	Complement	Action
0	0	Attribute unchanged
1	0	Attribute on
0	1	Attribute set to complement of default setting
1	1	Attribute off

wrap-flag

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the action to take if the text does not fit on the line. The **wrap-flag** argument is the address of an unsigned longword that contains the flag. Zero specifies no wrap (the default) while 1 specifies wrap.

char-set

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the default character set for all text in this virtual display. The **char-set** argument is the address of a longword that contains the character set code. At this time, the only valid value is SMG$C_ASCII, which is also the default.

DESCRIPTION

SMG$INSERT_LINE lets you insert a line into a virtual display at a location other than the first or last line. Existing lines are scrolled in the specified direction to create an open space. If you specify a **string** argument, that string is written in the space created; otherwise, the new line remains blank. If the string does not span the width of the display, it is padded with blanks.

If **wrap-flag** is set to 1 and the specified **string** is longer than the width of the virtual display, SMG$INSERT_LINE scrolls another line and writes the excess characters in the created space. If **wrap-flag** is 0, any excess characters are discarded. The virtual cursor remains at the character position following the last character written.

See SMG$PUT_LINE to add lines and scroll at the first or last line in a virtual display.

CONDITION VALUES RETURNED

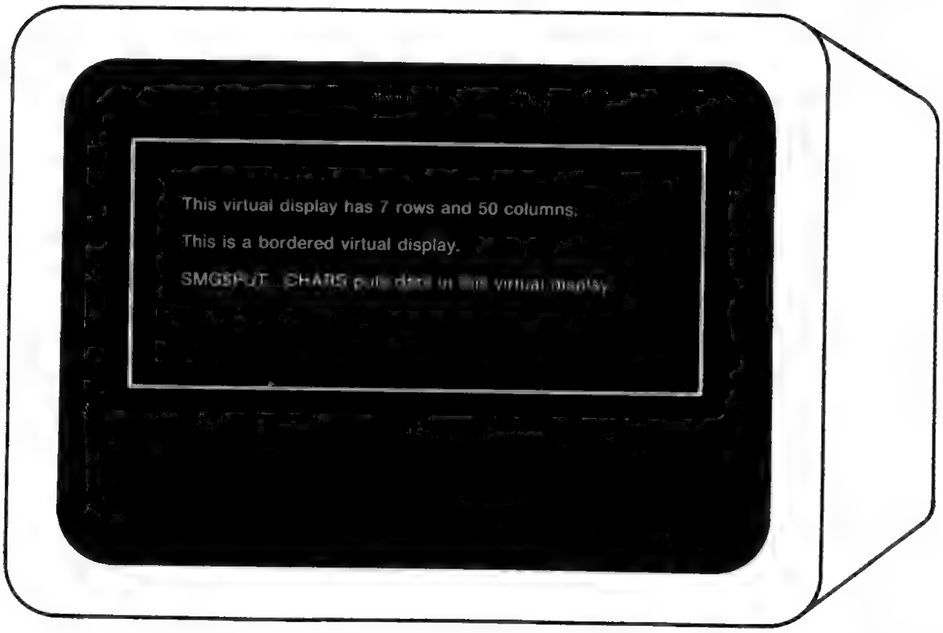
SS$_NORMAL	Normal successful completion.
SMG$_INVDIS_ID	Invalid **display-id**.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_INVROW	Invalid row.
SMG$_INVCOL	Invalid column.
SMG$_INVARG	Invalid argument. The specified direction is not up or down.

EXAMPLE

```
C+
C This FORTRAN example program demonstrates the use of SMG$INSERT_LINE.
C
C Include the SMG definitions. In particular, we want SMG$M_BORDER,
C SMG$M_UNDERLINE, and SMG$M_UP.
C-
        INCLUDE '($SMGDEF)'
        INTEGER SMG$CREATE_VIRTUAL_DISPLAY, SMG$CREATE_PASTEBOARD
        INTEGER SMG$PASTE_VIRTUAL_DISPLAY,  SMG$PUT_CHARS,
     1          SMG$ERASE_DISPLAY
        INTEGER DISPLAY1, PASTE1, ROWS, COLUMNS

C+
C Use SMG$CREATE_VIRTUAL_DISPLAY to create a virtual display
C with a border.
C-
        ROWS = 7
        COLUMNS = 50
        ISTATUS = SMG$CREATE_VIRTUAL_DISPLAY
     1          (ROWS, COLUMNS, DISPLAY1, SMG$M_BORDER)
        IF (.NOT. ISTATUS) WRITE (6, 900) 'SMG$CREATE_VIRTUAL_DISPLAY',
     1          ISTATUS

C+
C Call SMG$CREATE_PASTEBOARD to create the pasteboard.
C-
        ISTATUS = SMG$CREATE_PASTEBOARD (PASTE1)
        IF (.NOT. ISTATUS) WRITE (6, 900) 'SMG$CREATE_PASTEBOARD',
                ISTATUS

C+
C Use SMG$PUT_CHARS to put data in the virtual display.
C-
        DISTATUS = SMG$PUT_CHARS ( DISPLAY1,
     1          ' This virtual display has 7 rows and 50 columns.', 2, 1)
        ISTATUS = SMG$PUT_CHARS ( DISPLAY1,
     1          ' This is a bordered virtual display.', 4, 1)
        ISTATUS = SMG$PUT_CHARS ( DISPLAY1,
     1          ' SMG$PUT_CHARS puts data in this virtual display.', 6,
     1          1)

C+
C Paste the virtual display by calling SMG$PASTE_VIRTUAL_DISPLAY.
C-
        ISTATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DISPLAY1, PASTE1, 4, 15)
900     FORMAT (' Routine ', A, ' returned a status of ', Z8)
C+
C Call SMG$INSERT_LINE to add a line of text after line 6 and scroll
C the display. Also, underline the new characters.
C-
        ISTATUS = SMG$INSERT_LINE ( DISPLAY1, 7,
     1          'This is a new line.', SMG$M_UP, SMG$M_UNDERLINE)
        END
```

The initial output generated by this FORTRAN program is shown in Figure RTL–25.

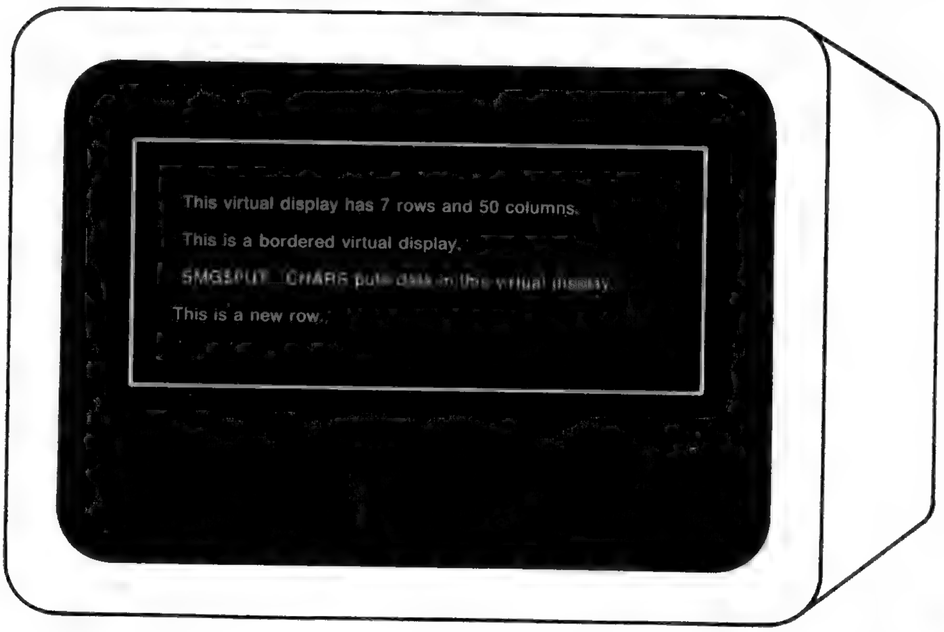
Figure RTL–25 Output Generated Before the Call to SMG$INSERT_LINE



ZK-4132-85

The output generated after the call to SMG$INSERT_LINE is shown in Figure RTL–26.

Figure RTL–26 Output Generated After the Call to SMG$INSERT_LINE



ZK-4131-85

SMG$INVALIDATE_DISPLAY—Mark a Display As Invalid

SMG$INVALIDATE_DISPLAY marks a display as invalid and causes the entire display to be redrawn.

FORMAT

SMG$INVALIDATE_DISPLAY *display-id*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENT

display-id
VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the virtual display affected. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

Display-id is returned by SMG$CREATE_VIRTUAL_DISPLAY.

DESCRIPTION

SMG$INVALIDATE_DISPLAY marks a display as invalid, and redraws the entire display. You would normally use this routine after you determine that output has been written to the display without benefit of the Screen Management Facility.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVDIS_ID	Invalid **display-id**.

SMG$LABEL_BORDER—Label A Virtual Display Border

SMG$LABEL_BORDER supplies a label for a virtual display's border.

FORMAT

SMG$LABEL_BORDER *display-id [,label-text] [,position] [,units] [,rendition_set] [,rendition-complement] [,char-set]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the virtual display affected. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

Display-id is returned by SMG$CREATE_VIRTUAL_DISPLAY.

label-text

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

The new label for this display's border. The **label-text** argument is the address of a descriptor pointing to the label text. If omitted, the display becomes unlabeled.

position

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies which of the display's borders contains the label. The **position** argument is the address of an unsigned longword that contains the position code.

Valid positions are as follows:

- SMG$K_TOP
- SMG$K_BOTTOM

- SMG$K_RIGHT
- SMG$K_LEFT

If this argument is omitted, the label is displayed on the top border.

units

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the character position at which the label begins within the border. The **units** argument is the address of a signed longword integer that contains the character position. If omitted, the label is centered in the specified border.

rendition-set

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Mask which denotes video attributes for the drawn line. The **rendition-set** argument is the address of an unsigned longword that contains a video attributes mask. Each bit attribute in this argument causes the corresponding attribute to be set in the display.

Video attributes which can be manipulated in this manner are as follows:

SMG$M_BLINK	Displays blinking characters
SMG$M_BOLD	Displays characters in higher-than-normal intensity
SMG$M_REVERSE	Displays characters in reverse video, that is, using the opposite default rendition of the virtual display
SMG$M_UNDERLINE	Displays underlined characters

If the same bit is set in both the **rendition-set** and **rendition-complement** arguments, the Screen Management Facility applies the **rendition-set** attribute followed by the **rendition-complement** attribute. Using these two arguments, the caller can exercise independent control over each attribute in a single call.

rendition-complement

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

A mask which denotes video attributes for the line drawn. The **rendition-complement** argument is the address of an unsigned longword that contains a video attributes mask. Each bit attribute in this argument causes the corresponding attribute to be set in the display. Video attributes which can be manipulated in this manner are the same as those for the **rendition-set** argument.

The following table shows the action taken by the Screen Management Facility for various combinations of **rendition-set** and **rendition-complement** attributes.

Set	Complement	Action
0	0	Attribute unchanged
1	0	Attribute on
0	1	Attribute set to complement of default setting
1	1	Attribute off

char-set

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the default character set for all text in this virtual display. The **char-set** argument is the address of an unsigned longword that contains the character set code. At this time, the only valid value is SMG$C_ASCII, which is also the default.

DESCRIPTION

SMG$LABEL_BORDER lets you specify text to label a virtual display. If the specified virtual display does not already have the border display attribute (SMG$M_BORDER), then this attribute is forced. If the label string is supplied, it replaces the current label text for this border. If you supply an empty (null) label string, the border becomes unlabeled. If the label text (as positioned within the border) does not fit within the border, this routine returns SMG$_INVARG.

Position and **units** together specify the starting position of the label text within a border. If **position** is omitted, the default is the top border. If **units** is omitted, this routine chooses a starting position so as to center the text either horizontally or vertically, depending on the implicit or explicit position argument. If both **position** and **units** are omitted, the text is centered in the top border.

Units specifies the label's starting row or column position in the border.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVDIS_ID	Invalid **display-id**.
SMG$_INVARG	Invalid argument. The combination of **position**, **units**, and **label-text** arguments resulted in a position outside the border area.
SMG$_WRONUMARG	Wrong number of arguments.

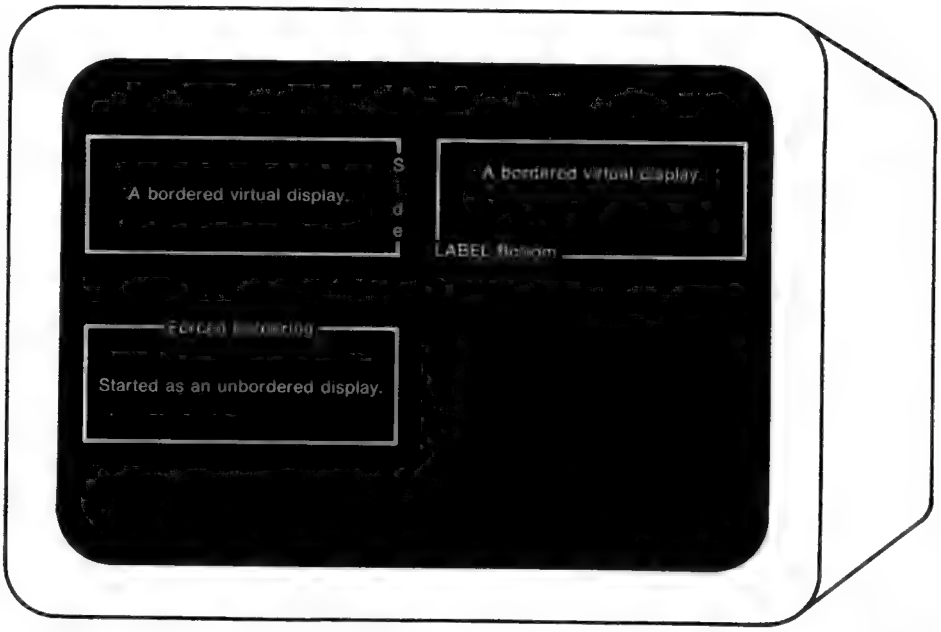
EXAMPLE

```
C+
C This FORTRAN example program demonstrates the use of SMG$LABEL_BORDER.
C-
C+
C Include the SMG definitions. In particular, we want SMG$M_BORDER,
C SMG$K_TOP, SMG$K_BOTTOM, and SMG$K_RIGHT.
C-
        INCLUDE '($SMGDEF)'
        INTEGER SMG$CREATE_VIRTUAL_DISPLAY, SMG$CREATE_PASTEBOARD
```

```
        INTEGER SMG$PASTE_VIRTUAL_DISPLAY,  SMG$PUT_CHARS
        INTEGER DISPLAY1, PASTE1
        INTEGER DISPLAY2, PASTE2
        INTEGER DISPLAY3, PASTE3, ROWS, COLUMNS
C+
C Call SMG$CREATE_VIRTUAL_DISPLAY to create virtual display number 1.
C Give it a border.
C-
        ROWS = 4
        COLUMNS = 30
        ISTATUS = SMG$CREATE_VIRTUAL_DISPLAY
     1          (ROWS, COLUMNS, DISPLAY1, SMG$M_BORDER)
C+
C Call SMG$CREATE_VIRTUAL_DISPLAY to create virtual display number 2.
C Give it a border.
C-
        ROWS = 3
        COLUMNS = 30
        ISTATUS = SMG$CREATE_VIRTUAL_DISPLAY
     1          (ROWS, COLUMNS, DISPLAY2, SMG$M_BORDER)
C+
C Create virtual display number 3. Do NOT give it a border.
C-
        ROWS = 4
        COLUMNS = 35
        ISTATUS = SMG$CREATE_VIRTUAL_DISPLAY
     1          (ROWS, COLUMNS, DISPLAY3)
C+
C Use SMG$CREATE_PASTEBOARD to create the pasteboard.
C-
        ISTATUS = SMG$CREATE_PASTEBOARD (PASTE1)
C+
C Call SMG$PUT_CHARS to put data into the virtual displays.
C-
        ISTATUS = SMG$PUT_CHARS ( DISPLAY1,
     1          ' A bordered virtual display.', 2, 1)
        ISTATUS = SMG$PUT_CHARS ( DISPLAY2,
     1          ' A bordered virtual display.', 1, 1)
        ISTATUS = SMG$PUT_CHARS ( DISPLAY3,
     1          ' Started as an unbordered display.', 2, 1)
C+
C Call SMG$LABEL_BORDER to label the virtual display borders.
C-
        ISTATUS = SMG$LABEL_BORDER ( DISPLAY1, 'Side', SMG$K_RIGHT)
        ISTATUS = SMG$LABEL_BORDER ( DISPLAY2, 'LABEL Bottom',
     1          SMG$K_BOTTOM, 1)
        ISTATUS = SMG$LABEL_BORDER ( DISPLAY3, 'Forced bordering ',
     1          SMG$K_TOP)
C+
C Call SMG$PASTE_VIRTUAL_DISPLAY to paste the virtual displays.
C-
        ISTATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DISPLAY1, PASTE1, 2, 10)
        ISTATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DISPLAY2, PASTE1, 2, 45)
        ISTATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DISPLAY3, PASTE1, 10, 5)
 900    FORMAT (' Routine ', A, ' returned a status of ', Z8)
        END
```

The output generated by this program is shown in Figure RTL–27.

Figure RTL-27 Output Generated by Program Calling SMG$LABEL_BORDER



ZK-4127-85

SMG$LIST_KEY_DEFS—List Key Definitions

SMG$LIST_KEY_DEFS returns the definition (equivalence string) associated with a specified key in a specified key table.

FORMAT

SMG$LIST_KEY_DEFS *key-table-id ,context [,key-name] [,if-state] [,attributes] [,equiv-string] [,state-string]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

key-table-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read-only**
mechanism: **by reference**

Specifies the key table from which you are extracting a key definition. The **key-table-id** argument is the address of an unsigned longword that contains the key table identifier.

Key-table-id is returned by the SMG$CREATE_KEY_TABLE routine.

context

VMS Usage: **context**
type: **longword integer (signed)**
access: **modify**
mechanism: **by reference**

Provides a means to extract a series of key definitions from a key table. The **context** argument is the address of a signed longword integer that contains the context variable. For the first call to this routine, you should set the **context** argument to zero.

Context is incremented by the SMG$LIST_KEY_DEFS routine so that the next call returns the next key definition.

key-name

VMS Usage: **char_string**
type: **character string**
access: **modify**
mechanism: **by descriptor**

Identifies the key whose value you are listing. The **key-name** argument is the address of a descriptor pointing to the key name.

if-state

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

Receives the state name which qualifies the next definition in the key table. The **if-state** argument is the address of a descriptor pointing to the storage into which the state name is written.

attributes

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Attributes of this key definition. The **attributes** argument is the address of an unsigned longword into which is written the key attributes.

Possible attributes are as follows:

SMG$V_KEY_NOECHO (Bit 0)	If set, this bit specifies that **equiv_string** is not to be echoed when this key is pressed; if clear, **equiv_string** is echoed. If SMG$V_KEY_TERMINATE is not set, SMG$V_KEY_NOECHO is ignored.
SMG$V_KEY_TERMINATE (Bit 1)	If set, this bit specifies that when this key is pressed (as qualified by **if-state**), the input line is complete and more characters should not be accepted. If clear, more characters may be accepted.
SMG$V_KEY_LOCKSTATE (Bit 2)	If set, and if **state-string** is specified, the state name specified by **state-string** remains the current state until explicitly changed by a subsequent keystroke whose definition includes a **state-string**. If clear, the state name specified by **state-string** remains in effect only for the next defined key stroke.
SMG$V_KEY_PROTECTED (Bit 3)	If set, this bit specifies that this key definition cannot be modified or deleted. If clear, the key definition can be modified or deleted.

The remaining bits are undefined.

equiv-string

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

The character string into which is written the equivalence string for the next key definition. The **equiv-string** argument is the address of a descriptor pointing to the storage into which the equivalence string is written.

state-string

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

A string into which is written the new state name, if any, which is set by the next key definition. The **state-string** argument is the address of a descriptor pointing to the storage into which the state name is written. If this key definition sets a state, the attributes flag SMG$V_KEY_SETSTATE is set.

DESCRIPTION

SMG$LIST_KEY_DEFS, when called repeatedly, lets you examine all the definitions in a key table.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVKTB_ID	Invalid **key-table-id**.
SMG$_NOMOREKEYS	No more keys in this table.

Any condition value returned by LIB$COPY_DXDX.

SMG$LOAD_KEY_DEFS—Load Key Definitions

SMG$LOAD_KEY_DEFS loads a file of key definitions (DEFINE/KEY commands) into a specified key table.

FORMAT

SMG$LOAD_KEY_DEFS *key-table-id ,filespec [,default-filespec] [,lognam-flag]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

key-table-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the key table into which you are loading key definitions. The **key-table-id** argument is the address of an unsigned longword that contains the key table identifier.

Key-table-id is returned by SMG$CREATE_KEY_TABLE.

filespec

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

String containing the file specification for the file of DEFINE/KEY commands. The **filespec** argument is the address of a descriptor pointing to the file specification.

default-filespec

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

String containing the default file specification for the file of DEFINE/KEY commands. The **default-filespec** argument is the address of a descriptor pointing to the default file specification. If omitted, the null string is used.

lognam-flag

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies whether **filespec** is to be treated as a logical name. The **lognam-flag** argument is the address of an unsigned longword that contains the flag. If set, **lognam-flag** specifies that filespec should be translated, but if this is not possible, that the null string be used.

DESCRIPTION

SMG$LOAD_KEY_DEFS opens and reads a file containing DEFINE/KEY commands and calls SMG$DEFINE_KEY for each command line in the file. Use of SMG$LOAD_KEY_DEFS requires that the calling program be run under the DCL command language interpreter. This routine signals any errors encountered while processing command lines.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_FILTOOLON	File specification is too long (over 255 characters).

Any condition values returned by SMG$DEFINE_KEY.

Any condition values returned by $OPEN.

SMG$MOVE_VIRTUAL_DISPLAY—Move Virtual Display

SMG$MOVE_VIRTUAL_DISPLAY relocates a virtual display on a pasteboard and preserves the pasting order.

FORMAT

SMG$MOVE_VIRTUAL_DISPLAY
display-id ,pasteboard-id ,pasteboard-row ,pasteboard-column [,top-display-id]

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

display-id

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Specifies the virtual display to be moved. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

Display-id is returned by SMG$CREATE_VIRTUAL_DISPLAY.

pasteboard-id

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Specifies the pasteboard on which the movement is to take place. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

Pasteboard-id is returned by SMG$CREATE_PASTEBOARD.

pasteboard-row

VMS Usage:	**longword_signed**
type:	**longword integer (signed)**
access:	**read only**
mechanism:	**by reference**

Specifies the row of the pasteboard that is to contain row 1 of the specified virtual display. The **pasteboard-row** argument is the address of a signed longword integer that contains the row number.

pasteboard-column

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the column of the pasteboard that is to contain column 1 of the specified virtual display. The **pasteboard-column** argument is the address of a signed longword integer that contains the column number.

top-display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Identifier of the virtual display under which the unpasted **display-id** will be pasted. The **top-display-id** argument is the address of an unsigned longword containing the specified virtual display identifier. Note that the use of the **top-display-id** argument is only valid when the virtual display specified by **display-id** is not currently pasted and the virtual display specified by **top-display-id** is pasted.

DESCRIPTION

SMG$MOVE_VIRTUAL_DISPLAY moves a pasted virtual display from its current position to the specified position and preserves the pasting order. If the display being moved is not currently pasted, SMG$MOVE_VIRTUAL_DISPLAY presents the user with 2 options. By default, SMG$MOVE_VIRTUAL_DISPLAY will paste the display at the top of the pasting order in the position specified.

If, however, the optional argument **top-display-id** is specified, SMG$MOVE_VIRTUAL_DISPLAY pastes the virtual display being moved under the virtual display specified by **top-display-id**. In this case, the virtual display specified by **top-display-id** must already be pasted.

Note that a display cannot be moved from one pasteboard to another. However, the **pasteboard-id** is required because a given virtual display may be pasted to any number of pasteboards.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVDIS_ID	Invalid **display-id**.
SMG$_INVPAS_ID	Invalid **pasteboard-id**.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_ILLBATFNC	Display is being batched; illegal operation.

EXAMPLE

Refer to the FORTRAN example shown in the SMG$REPASTE_VIRTUAL_DISPLAY routine.

SMG$PASTE_VIRTUAL_DISPLAY Paste Virtual Display

SMG$PASTE_VIRTUAL_DISPLAY pastes a virtual display to a pasteboard.

FORMAT

SMG$PASTE_VIRTUAL_DISPLAY *display-id ,pasteboard-id ,pasteboard-row ,pasteboard-column [,top-display-id]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the virtual display to be pasted. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

Display-id is returned by SMG$CREATE_VIRTUAL_DISPLAY.

pasteboard-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the pasteboard to which the display is to be pasted. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

pasteboard-row

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the row of the pasteboard that is to contain row 1 of the specified virtual display. The **pasteboard-row** argument is the address of a signed longword integer that contains the row number.

pasteboard-column

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the column of the pasteboard that is to contain column 1 of the specified virtual display. The **pasteboard-column** argument is the address of a signed longword integer that contains the column number.

top-display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Identifier of the virtual display under which to paste **display-id**. The optional **top-display-id** argument is the address of an unsigned longword containing this identifier. Note that the virtual display specified by **top-display-id** must already be pasted.

DESCRIPTION

SMG$PASTE_VIRTUAL_DISPLAY places a display on a pasteboard and makes the display visible. If, however, the optional argument **top-display-id** is specified, SMG$PASTE_VIRTUAL_DISPLAY pastes the virtual display being pasted under the virtual display specified by **top-display-id**. In this case, the virtual display specified by **top-display-id** must already be pasted.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVDIS_ID	Invalid **display-id**.
SMG$_INVPAS_ID	Invalid **pasteboard-id**.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_ILLBATFNC	Display is being batched; illegal operation.

EXAMPLE

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
     C           CREPAS    EXTRN'SMG$CREATE_PASTEBOARD'
     C           CREDIS    EXTRN'SMG$CREATE_VIRTUAL_DISPLAY'
     C           PUTCHA    EXTRN'SMG$PUT_CHARS'
     C           PASDIS    EXTRN'SMG$PASTE_VIRTUAL_DISPLAY'
     C                     Z-ADD0         ZERO    90
     C                     Z-ADD1         LINCOL  90
     C                     Z-ADD2         LINE    90
     C                     Z-ADD5         COLUMN  90
     C                     MOVE 'Menu'    OUT      4
     C* Create the pasteboard.
     C                     CALL CREPAS
     C                     PARM           PASTID  90 WL
     C                     PARMV          ZERO
     C                     PARM           HEIGHT  90 WL
     C                     PARM           WIDTH   90 WL
     C* Create the virtual display.
     C                     CALL CREDIS
     C                     PARM           HEIGHT     RL
     C                     PARM           WIDTH      RL
```

```
     C                       PARM           DISPID  90 WL
     C* Output the 'Menu'.
     C                       CALL PUTCHA
     C                       PARM           DISPID     RL
     C                       PARMD          OUT
     C                       PARM           LINE       RL
     C                       PARM           COLUMN     RL
     C* Paste the virtual display.
     C                       CALL PASDIS
     C                       PARM           DISPID     RL
     C                       PARM           PASTID     RL
     C                       PARM           LINCOL     RL
     C                       PARM           LINCOL     RL
     C                       SETON                     LR
The RPG II program above displays 'Menu' beginning at line 2 column 5.
```

This RPG II program calls several SMG$ routines. For another example of how to call SMG$PASTE_VIRTUAL_DISPLAY, see the RPG II example in the description of SMG$CREATE_PASTEBOARD.

SMG$POP_VIRTUAL_DISPLAY—Delete a Series of Virtual Displays

SMG$POP_VIRTUAL_DISPLAY deletes a specified virtual display and all displays that were pasted on the specified pasteboard after the specified virtual display.

FORMAT

SMG$POP_VIRTUAL_DISPLAY *display-id ,pasteboard-id*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

display-id

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Specifies the lowest (first) virtual display to be deleted. The **display-id** argument is the address of an unsigned longword that contains the display identifier. All displays that are higher in the pasting order (that is, all displays that were pasted after the specified display) are deleted as well.

Display-id is returned by SMG$CREATE_VIRTUAL_DISPLAY.

pasteboard-id

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Specifies the pasteboard on which the display deletions take place. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

DESCRIPTION

SMG$POP_VIRTUAL_DISPLAY deletes (not merely unpastes) one or more displays from the specified pasteboard, starting with the display specified and including all displays that are higher in the pasting order (that is, all displays that were pasted after the specified display).

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVDIS_ID	Invalid **display-id**.
SMG$_INVPAS_ID	Invalid **pasteboard-id**.
SMG$_WRONUMARG	Wrong number of arguments.

SMG$PUT_LINE—Write Line to Virtual Display

SMG$PUT_LINE writes a line of text to a virtual display.

FORMAT

SMG$PUT_LINE *display-id ,text [,line-advance] [,rendition-set] [,rendition-complement] [,wrap-flag] [,char-set] [,direction]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the virtual display affected. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

Display-id is returned by SMG$CREATE_VIRTUAL_DISPLAY.

text

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

The characters to be written to the virtual display. The **text** argument is the address of a descriptor pointing to the text.

line-advance

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the number of lines to advance after output. The **line-advance** argument is the address of a signed longword integer that contains the number of lines to advance. The default is 1.

rendition-set

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

A mask that denotes video attributes for the drawn line. The **rendition-set** argument is the address of an unsigned longword that contains a video attributes mask. Each bit attribute in this argument causes the corresponding attribute to be set in the display.

Video attributes that can be manipulated in this manner are as follows:

SMG$M_BLINK	Displays blinking characters.
SMG$M_BOLD	Displays characters in higher-than-normal intensity.
SMG$M_REVERSE	Displays characters in reverse video, that is, using the opposite default rendition of the virtual display.
SMG$M_UNDERLINE	Displays underlined characters.

If the same bit is set in both the **rendition-set** and **rendition-complement** arguments, the Screen Management Facility applies the **rendition-set** attribute followed by the **rendition-complement** attribute. Using these two arguments, the caller can exercise independent control over each attribute in a single call.

rendition-complement

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

A mask that denotes video attributes for the line drawn. The **rendition-complement** argument is the address of an unsigned longword that contains a video attributes mask. Each bit attribute in this argument causes the corresponding attribute to be set in the display. Video attributes that can be manipulated in this manner are the same as those for the **rendition-set** argument.

The following table shows the action taken by the Screen Management Facility for various combinations of **rendition-set** and **rendition-complement** attributes.

Set	Complement	Action
0	0	Attribute unchanged.
1	0	Attribute on.
0	1	Attribute set to complement of default setting.
1	1	Attribute off.

wrap-flag

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the action to take if the text does not fit on the line. The **wrap-flag** argument is the address of an unsigned longword that contains the flag. Zero specifies no wrap (the default) while 1 specifies wrap.

char-set

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the default character set for all text in this virtual display. The **char-set** argument is the address of an unsigned longword that contains the character set code. At this time, the only valid value is SMG$C_ASCII, which is also the default.

direction

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the direction to scroll, if scrolling is necessary. The **direction** argument is the address of an unsigned longword that contains the direction code. Valid values are SMG$M_UP and SMG$M_DOWN. SMG$M_UP is the default.

DESCRIPTION

SMG$PUT_LINE writes lines of text to the virtual display, beginning at the current line. Once text reaches the bottom or top line (depending on the scrolling direction), subsequent calls to this routine cause the display to scroll. SMG$PUT_LINE writes out the entire line, starting at the current virtual cursor position. If the caller's text does not span the entire line, the line is padded with blanks.

If **wrap-flag** is set, lines are scrolled **line-advance** times to make room for the overflow characters in the "next" line. The "next" line is determined by the scrolling **direction**. If **wrap-flag** is clear, excess characters are discarded.

Following a call to SMG$PUT_LINE, the virtual cursor position is set to column 1 of the next line where output should occur. The next line where output should occur is determined by the **line-advance** argument; **line-advance** defaults to 1 so that subsequent calls to SMG$PUT_LINE will not cause overprinting. Other SMG$ procedures that can be used to write lines of text to a virtual display are SMG$PUT_LINE_WIDE and SMG$PUT_LINE_HIGHWIDE.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVDIS_ID	Invalid **display-id**.
SMG$_WRONUMARG	Wrong number of arguments.
LIB$_INVSTRDES	Invalid string descriptor.

EXAMPLES

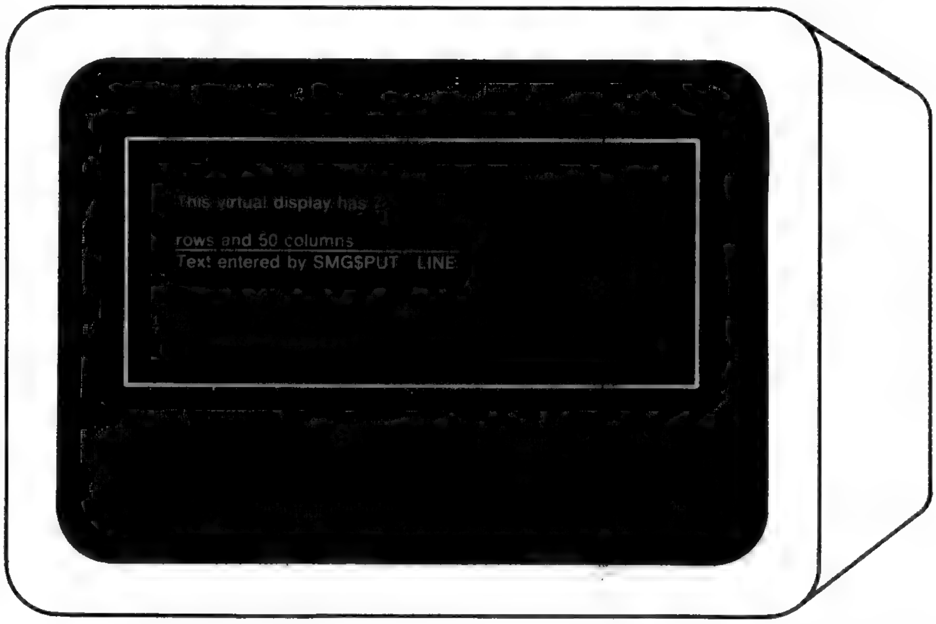
1

```
C+
C This FORTRAN example program demonstrates the use of SMG$PUT_LINE.
C-
C+
C Include the SMG definitions. In particular, we want SMG$M_BORDER and
C SMG$M_UNDERLINE.
C-
```

```
        IMPLICIT INTEGER (A-Z)
        INCLUDE '($SMGDEF)'
        CHARACTER*30 TEXT(3)
C+
C Create a virtual display with a border.
C-
        ROWS = 7
        COLUMNS = 50
        STATUS = SMG$CREATE_VIRTUAL_DISPLAY
     1  (ROWS, COLUMNS, DISPLAY1, SMG$M_BORDER)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Create the pasteboard.
C-
        STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Put data in the virtual display.
C-
        TEXT(1) = 'This virtual display has 7'
        TEXT(2) = 'rows and 50 columns.'
        TEXT(3) = 'Text entered by SMG$PUT_LINE.'
C+
C After the first line of text is printed, call SMG$PUT_LINE to
C advance two lines.
C-
        STATUS = SMG$PUT_LINE ( DISPLAY1, TEXT(1), 2)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Now, use SMG$PUT_LINE to underline the next line of text.
C Notice that 30 characters are being underlined. Advance 1
C line of text after displaying the line.
C-
        STATUS = SMG$PUT_LINE ( DISPLAY1, TEXT(2), 1, SMG$M_UNDERLINE)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Display the third line of text.
C-
        STATUS = SMG$PUT_LINE ( DISPLAY1, TEXT(3))
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Paste the virtual display.
C-
        STATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DISPLAY1, PASTE1, 4, 15)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        END
```

The output generated by this FORTRAN program is shown in Figure RTL–29.

Figure RTL-29 Output Generated by FORTRAN Program Calling SMG$PUT_LINE



ZK-4135-85

The following program illustrates the use of the new **direction** argument to SMG$PUT_LINE. This new capability has made the Screen Management Routine SMG$PUT_WITH_SCROLL obsolete.

2

```
C+
C This FORTRAN example program demonstrates the use of the DIRECTION
C parameter in the SMG$PUT_LINE routine.
C
C The DIRECTION parameter in SMG$PUT_LINE makes SMG$PUT_WITH_SCROLL
C an obsolete routine. This example is the same as the SMG$PUT_WITH_SCROLL,
C except that the calls to SMG$PUT_WITH_SCROLL have been replace by calls
C to SMG$PUT_LINE.
C-
        PARAMETER       SMG$M_UP = 1
        PARAMETER       SMG$M_DOWN = 2
        PARAMETER       SMG$M_BOLD = 1
        PARAMETER       SMG$M_REVERSE = 2
        PARAMETER       SMG$M_BLINK = 4
        PARAMETER       SMG$M_UNDERLINE = 8

        IMPLICIT INTEGER*4 (A-Z)
C+
C Call SMG$CREATE_PASTEBOARD to establish the terminal screen
C as a pasteboard.
C-
        STATUS = SMG$CREATE_PASTEBOARD (NEW_PID)
        IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Using SMG$CREATE_VIRTUAL_DISPLAY, establish a virtual display region.
C-
        STATUS = SMG$CREATE_VIRTUAL_DISPLAY (5,80,DISPLAY_ID)
        IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Paste the virtual display to the screen, starting at
C row 10, column 15 by calling SMG$PASTE_VIRTUAL_DISPLAY.
```

```
C-
      STATUS = SMG$PASTE_VIRTUAL_DISPLAY(DISPLAY_ID,NEW_PID,10,15)
      IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Define a scrolling region through a call to
C SMG$SET_DISPLAY_SCROLL_REGION.
C-
      STATUS = SMG$SET_DISPLAY_SCROLL_REGION(DISPLAY_ID,1,5)
      IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Call SMG$PUT_LINE and SMG$ERASE_LINE to write three
C scrolling lines to the screen.  The first line will be underlined,
C the second blinking, and the third in reverse video.
C-
      DO I = 1,10
      IF ((I/2) + (I/2) .EQ. I) THEN
              DIR = SMG$M_UP
      ELSE
              DIR = SMG$M_DOWN
      ENDIF
      STATUS = SMG$PUT_LINE (DISPLAY_ID,
     1     'This line is  underlined',,SMG$M_UNDERLINE,,,,DIR)
      IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
      STATUS = SMG$ERASE_LINE(DISPLAY_ID)
      IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
      STATUS = SMG$PUT_LINE (DISPLAY_ID,'This line is blinking', ,
     1     SMG$M_BLINK,,,,DIR)
      IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
      STATUS = SMG$ERASE_LINE (DISPLAY_ID)
      IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
      STATUS = SMG$PUT_LINE (DISPLAY_ID,'This line is reverse
     1     video',,SMG$M_REVERSE,,,,DIR)
      IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
      STATUS = SMG$ERASE_LINE (DISPLAY_ID)
      IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
      ENDDO
      END
```

SMG$PUT_LINE_HIGHWIDE Write Double High and Double Wide Line

SMG$PUT_LINE_HIGHWIDE writes lines with double high and double wide characters.

FORMAT

SMG$PUT_LINE_HIGHWIDE
display-id ,text [,line-adv] [,rendition-set] [,rendition-complement] [,wrap-flag] [,char-set]

RETURNS

VMS Usage: **cond_value**
type: **longword integer (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display identifier. The **display-id** argument is the address of an unsigned longword that contains the display identifier of the virtual display.

text

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Text output. The **text** argument is the address of the descriptor pointing to the output string.

line-adv

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Lines to advance. The **line-adv** argument is the address of a signed longword that contains the number of lines to advance after the output. This argument is optional.

rendition-set

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Set rendition. The **rendition-set** argument is the address of an unsigned longword that contains attribute information. Each 1-bit attribute in this argument causes the corresponding attribute to be set in the display (see below for list of the attributes that can be set).

rendition-complement

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Complement rendition. The **rendition-complement** argument is the address of an unsigned longword that contains attribute information. Each 1-bit attribute in this argument causes the corresponding attribute to be complemented in the display (see below for a list of the attributes that can be set).

If the same bit is specified in both the **rendition-set** argument and in the **rendition-complement** argument, the application is **rendition-set** followed by **rendition-complement**. Using these two arguments together, the caller can exercise arbitrary and independent control over each attribute on a single call. On an attribute-by-attribute basis the following transformations can be caused:

Set	Complement	Action
0	0	Attribute unchanged.
1	0	Attribute set to "on."
0	1	Attribute set to complement of current setting.
1	1	Attribute set to "off."

Attributes that can be manipulated in this manner are:

SMG$M_BLINK	Displays characters blinking.
SMG$M_BOLD	Displays characters in higher-than-normal intensity.
SMG$M_REVERSE	Displays characters in reverse video—that is, using the opposite default rendition of the virtual display.
SMG$M_UNDERLINE	Displays characters underlined.

wrap-flag

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

The value used to determine if wrapping is enabled or not. The **wrap-flag** argument is an unsigned longword that contains this value. A 0 means no wrap; a 1 means wrap enabled. If the value is omitted, no wrap is the default.

char-set

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the default character set for all text in this virtual display. The **char-set** argument is the address of an unsigned longword that contains the character-set code. At this time, the only valid value is SMG$C_ASCII, which is also the default.

DESCRIPTION

This routine is used to write lines with double-high and double-wide characters to the virtual display. SMG$PUT_LINE_HIGHWIDE writes from the current virtual cursor position to the end of the line. If the caller's text does not span to the end of the line, blank spaces are added.

Treatment of text that exceeds the rightmost bounds of the display depends on the **wrap-flag** argument. If the **wrap-flag** is set, lines are scrolled **line-adv** times to make room for the overflow characters in the "next" line. If wrap is off, overflow characters are lost.

Following a call to SMG$PUT_LINE_HIGHWIDE, the virtual cursor position is set to column 1 of the next line where output should occur. The next line where output should occur is determined by the **line-adv** argument. **Line-adv** defaults to 2 so that subsequent calls to SMG$PUT_LINE_HIGHWIDE will not cause overprinting.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_WRONUMARG	Wrong number (or combination of) arguments.
LIB$_INVSTRDES	Invalid string descriptor.

SMG$PUT_LINE_WIDE—Write Double-Width Line

SMG$PUT_LINE_WIDE writes a line of double-width text to a virtual display.

FORMAT

SMG$PUT_LINE_WIDE *display-id ,text [,line-advance] [,rendition-set] [,rendition-complement] [,wrap-flag] [,char-set]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the virtual display affected. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

Display-id is returned by SMG$CREATE_VIRTUAL_DISPLAY.

text

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Characters to be written to the virtual display. The **text** argument is the address of a descriptor pointing to the text.

line-advance

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the number of lines to advance after output. The **line-advance** argument is the address of a signed longword integer that contains the number of lines to advance.

rendition-set

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Mask that denotes video attributes for the drawn line. The **rendition-set** argument is the address of an unsigned longword that contains a video attributes mask. Each bit attribute in this argument causes the corresponding attribute to be set in the display.

Video attributes that can be manipulated in this manner are as follows:

SMG$M_BLINK	Displays blinking characters
SMG$M_BOLD	Displays characters in higher-than-normal intensity
SMG$M_REVERSE	Displays characters in reverse video — that is, using the opposite default rendition of the virtual display
SMG$M_UNDERLINE	Displays underlined characters

If the same bit is set in both the **rendition-set** and **rendition-complement** arguments, the Screen Management Facility applies the **rendition-set** attribute followed by the **rendition-complement** attribute. Using these two arguments, the caller can exercise independent control over each attribute in a single call.

rendition-complement

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Mask that denotes video attributes for the line drawn. The **rendition-complement** argument is the address of an unsigned longword that contains a video attributes mask.

Each bit attribute in this argument causes the corresponding attribute to be set in the display. Video attributes that can be manipulated in this manner are the same as those for the **rendition-set** argument.

The following table shows the action taken by the Screen Management Facility for various combinations of **rendition-set** and **rendition-complement** attributes.

Set	Complement	Action
0	0	Attribute unchanged
1	0	Attribute on
0	1	Attribute set to complement of default setting
1	1	Attribute off

wrap-flag

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the action to take if the text does not fit on the line. The **wrap-flag** argument is the address of an unsigned longword that contains the flag. 0 specifies no wrap (the default) while 1 specifies wrap.

char-set

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the default character set for all text in this virtual display. The **char-set** argument is the address of an unsigned longword that contains the character set code. At this time, the only valid value is SMG$C_ASCII, which is also the default.

DESCRIPTION

SMG$PUT_LINE_WIDE write lines of double-width text to the virtual display. SMG$PUT_LINE_WIDE writes out the entire line, starting at the current virtual cursor position. If the caller's text does not span the entire line, the line is filled with blanks.

If **wrap-flag** is set, lines are scrolled **line-advance** times to make room for the overflow characters in the "next" line. If **wrap-flag** is clear, excess characters are discarded.

Following a call to SMG$PUT_LINE_WIDE, the virtual cursor position is set to column 1 of the next line where output should occur. The next line where output should occur is determined by the **line-advance** argument; **line-advance** defaults to 1 so that subsequent calls to SMG$PUT_LINE will not cause overprinting.

Other procedures that may be used to write text to a virtual display are SMG$PUT_LINE and SMG$PUT_LINE_HIGHWIDE.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVDIS_ID	Invalid **display-id**.
SMG$_WRONUMARG	Wrong number of arguments.
LIB$_INVSTRDES	Invalid string descriptor.

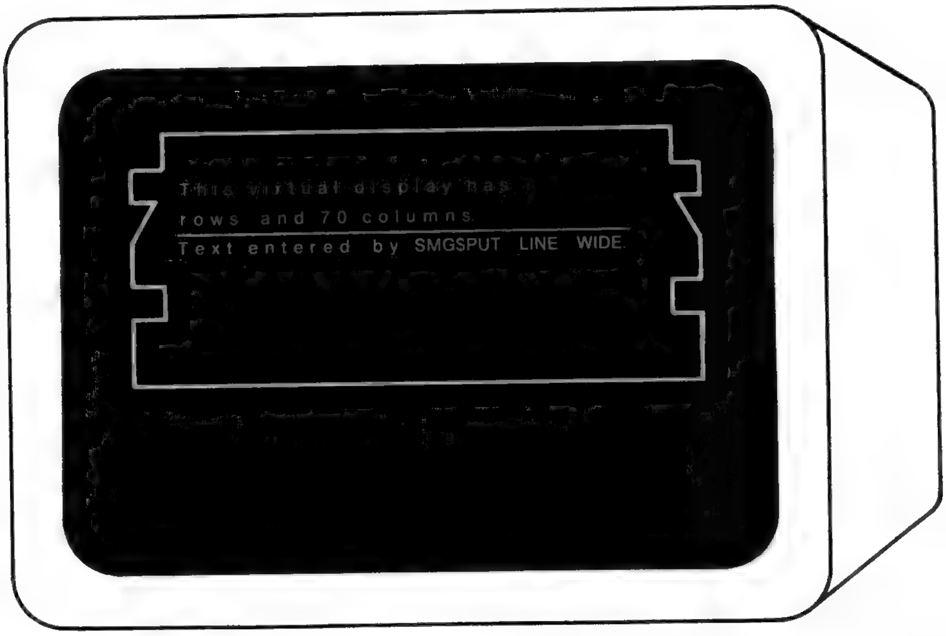
EXAMPLE

```
C+
C This FORTRAN example program demonstrates the use of
C SMG$PUT_LINE_WIDE.
C
C Include the SMG definitions. In particular, we want SMG$M_BORDER and
C SMG$M_UNDERLINE.
C-
        INCLUDE '($SMGDEF)'
        INTEGER SMG$CREATE_VIRTUAL_DISPLAY, SMG$CREATE_PASTEBOARD
        INTEGER SMG$PASTE_VIRTUAL_DISPLAY,  SMG$PUT_LINE_WIDE
```

```
        INTEGER DISPLAY1, PASTE1, ROWS, COLUMNS, STATUS
        CHARACTER*34 TEXT(3)
C+
C Create a virtual display with a border by calling
C SMG$CREATE_VIRTUAL_DISPLAY.
C-
        ROWS = 7
        COLUMNS = 70
        STATUS = SMG$CREATE_VIRTUAL_DISPLAY
     1          (ROWS, COLUMNS, DISPLAY1, SMG$M_BORDER)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Call SMG$CREATE_PASTEBOARD to create the pasteboard.
C-
        STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Use SMG$PUT_LINE to put data in the virtual display.
C-
        TEXT(1) = 'This virtual display has 7'
        TEXT(2) = 'rows and 70 columns.'
        TEXT(3) = 'Text entered by SMG$PUT_LINE_WIDE.'
C+
C After the first line of text is printed, advance two lines.
C-
        STATUS = SMG$PUT_LINE_WIDE ( DISPLAY1, TEXT(1), 2)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Underline the next line of text. Notice that 34 characters are being
C underlined. Advance 1 line of text after displaying the line.
C-
        STATUS = SMG$PUT_LINE_WIDE ( DISPLAY1, TEXT(2), 1,
     1          SMG$M_UNDERLINE)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Display the third line of text.
C-
        STATUS = SMG$PUT_LINE_WIDE ( DISPLAY1, TEXT(3))
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Paste the virtual display using SMG$PASTE_VIRTUAL_DISPLAY.
C-
        STATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DISPLAY1, PASTE1, 4, 5)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        END
```

The output generated by this FORTRAN program is shown in Figure RTL-30.

Figure RTL-30 Output Generated by FORTRAN Program Calling SMG$PUT_LINE_WIDE



SMG$PUT_PASTEBOARD Output Pasteboard via Routine

SMG$PUT_PASTEBOARD accesses the contents of a pasteboard.

FORMAT

SMG$PUT_PASTEBOARD *pasteboard-id ,p-rtn ,p-prm ,p-ff-flag*

RETURNS

VMS Usage: **cond_value**
type: **longword integer (signed)**
access: **write only**
mechanism: **by value**

ARGUMENTS

pasteboard-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Pasteboard identifier. The **pasteboard-id** argument is the address of an unsigned longword containing the pasteboard identifier.

p-rtn

VMS Usage: **longword_unsigned**
type: **procedure entry mask**
access: **read only**
mechanism: **by reference**

Pasteboard routine. The **p-rtn** argument is the address of the routine to be called.

p-prm

VMS Usage: **user_arg**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Pasteboard argument. The **p-prm** argument is a user-specified argument to be passed to the action routine. If omitted, a 0 will be passed as the user argument.

p-ff-flag

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

A flag (0 or 1). If 1, then the first line passed to the action routine will be a form feed. If not specified, then no form-feed line will be sent.

DESCRIPTION

The SMG$PUT_PASTEBOARD routine accesses the contents of a pasteboard. The caller specifies an action routine that will be called once for each line in the pasteboard. The action routine will be passed a descriptor for that line followed by a user-specified argument.

descriptor for line
p-prm argument

ZK-4991-86

CONDITION VALUES SIGNALED

SS$_NORMAL	Normal successful completion.
Other	Error return passed back by an action routine.

SMG$PUT_VIRTUAL_DISPLAY_ENCODED Write Encoded String To Display

SMG$PUT_VIRTUAL_DISPLAY_ENCODED lets you write a string that has multiple video renditions to a virtual display.

FORMAT

SMG$PUT_VIRTUAL_DISPLAY_ENCODED *display-id ,encoded-length ,encoded-text [,line-number] [,column-number] [,placeholder-arg] [,char-set]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the virtual display affected. The **display-id** argument is the address of a signed longword integer that contains the display identifier.

Display-id is returned by SMG$CREATE_VIRTUAL_DISPLAY.

encoded-length

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Length of the encoded string. The **encoded-length** argument is the address of an unsigned longword that contains the length of the encoded string.

encoded-text

VMS Usage: **address**
type: **unspecified**
access: **read only**
mechanism: **by reference**

A data structure or record passed by reference. The **encoded-text** argument is the address of the data.

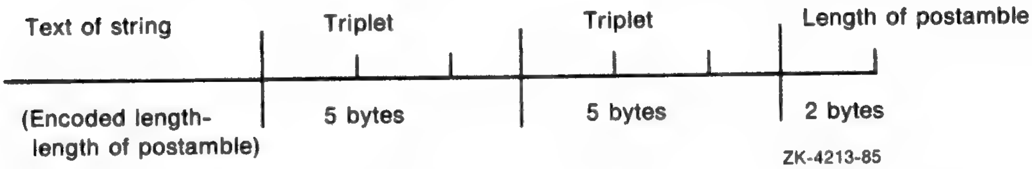
Encoded-text has three parts:

- The text to be displayed.

- One or more "triplets," each specifying: 1) the starting position of a field within the text, 2) the length of the field, and 3) the video rendition for that field.
- An unsigned word integer specifying the number of bytes in **encoded-text** that are not part of the text to be displayed. That is, the number of bytes in all triplets, plus two bytes for the unsigned word integer.

Figure RTL–31 shows the format of the **encoded-text** argument.

Figure RTL–31 Format of Encoded Text



Each triplet requires five bytes:

- Two bytes for an unsigned word integer specifying the starting position of the field within the text. The first position is numbered 1.
- Two bytes for an unsigned word integer specifying the number of bytes in the field.
- One byte specifying the video attributes for the field.

Each triplet is treated as a self-contained unit, and video attributes are applied in the order specified. Thus, if one triplet specifies underlining for character positions 1 through 5, and another triplet specifies bolding for character positions 3 through 5, bolding overwrites underlining for character positions 3 through 5. To specify more than one video attribute for a field, use the bitwise OR of the video attribute codes. Any text not specified in a triplet is displayed with no video attributes.

line-number

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the line at which output begins. The **line-number** argument is the address of a signed longword integer that contains the line number. If **line-number** is omitted or if it is equal to zero, output begins on the current line.

column-number

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the column at which output begins. The **column-number** argument is the address of a signed longword integer that contains the column number. If **column-number** is omitted or if it is equal to zero, output begins on the current column.

placeholder-arg

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Reserved placeholder. The **placeholder-arg** argument is a reserved placeholder.

char-set

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the default character set for all text in this virtual display. The **char-set** argument is the address of an unsigned longword that contains the character set code. At this time, the only valid value is SMG$C_ASCII, which is also the default.

DESCRIPTION

SMG$PUT_VIRTUAL_DISPLAY_ENCODED writes the encoded string to the specified virtual display. If line-number and column-number are both omitted, the string is written starting at the current virtual cursor position. If the text does not span the entire line, the remaining space is filled with blanks.

This routine is useful for writing strings in which some but not all characters have video renditions, or where different video renditions are desired for the same string.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVARG	Invalid argument or combination of arguments. For example, one of the attribute triplets may specify a starting byte and a number of bytes that extend beyond the length of the string.
SMG$_INVDIS_ID	Invalid **display-id**.
LIB$_INSVIRMEM	Insufficient virtual memory.

SMG$PUT_WITH_SCROLL—Write Text and Scroll

SMG$PUT_WITH_SCROLL writes a line of text to a virtual display and scrolls the display if necessary.

FORMAT

SMG$PUT_WITH_SCROLL *display-id [,text] [,direction] [,rendition-set] [,rendition-complement] [,wrap-flag] [,char-set]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the virtual display affected. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

Display-id is returned by SMG$CREATE_VIRTUAL_DISPLAY.

text

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

The text to be displayed. The **text** argument is the address of a descriptor pointing to the text.

direction

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the direction to scroll, if scrolling is necessary. The **direction** argument is the address of an unsigned longword that contains the direction code. Valid values are SMG$M_UP and SMG$M_DOWN. SMG$M_UP is the default.

rendition-set

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Mask that denotes video attributes for the drawn line. The **rendition-set** argument is the address of an unsigned longword that contains a video attributes mask. Each bit attribute in this argument causes the corresponding attribute to be set in the display.

Video attributes that can be manipulated in this manner are as follows:

SMG$M_BLINK	Displays blinking characters
SMG$M_BOLD	Displays characters in higher-than-normal intensity
SMG$M_REVERSE	Displays characters in reverse video — that is, using the opposite default rendition of the virtual display
SMG$M_UNDERLINE	Displays underlined characters

If the same bit is set in both the **rendition-set** and **rendition-complement** arguments, the Screen Management Facility applies the **rendition-set** attribute followed by the **rendition-complement** attribute. Using these two arguments, the caller can exercise independent control over each attribute in a single call.

rendition-complement

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Mask that denotes video attributes for the line drawn. The **rendition-complement** argument is the address of an unsigned longword that contains a video attributes mask.

Each bit attribute in this argument causes the corresponding attribute to be set in the display. Video attributes that can be manipulated in this manner are the same as those for the **rendition-set** argument.

The following table shows the action taken by the Screen Management Facility for various combinations of **rendition-set** and **rendition-complement** attributes.

Set	Complement	Action
0	0	Attribute unchanged
1	0	Attribute on
0	1	Attribute set to complement of default setting
1	1	Attribute off

wrap-flag

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the action to take if the text does not fit on the line. The **wrap-flag** argument is the address of an unsigned longword that contains the flag. No wrap (the default) is 0, while 1 is wrap.

char-set

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the default character set for all text in this virtual display. The **char-set** argument is the address of an unsigned longword that contains the character set code. At this time, the only valid value is SMG$C_ASCII, which is also the default.

DESCRIPTION

SMG$PUT_WITH_SCROLL writes text to the specified virtual display beginning at the current line. Once text reaches the bottom or top line (depending on the scrolling direction), subsequent calls to this routine cause the display to scroll. If you do not supply any text, this routine opens a blank line.

If **wrap-flag** is set, and the text provided exceeds the rightmost boundary of the display, another line is written and the display scrolled, if necessary. If **wrap-flag** is off, overflow characters are discarded.

Note that you cannot use SMG$PUT_WITH_SCROLL to insert a line into the middle of a virtual display and scroll the displaced lines. To do this you should first redefine the display's scrolling region with SMG$SCROLL_DISPLAY_AREA, then write the line to the beginning of the new scrolling region in the virtual display.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVDIS_ID	Invalid **display-id**.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_INVARG	Invalid argument, for example, a direction other than up or down.
LIB$_INVSTRDES	Invalid string descriptor.

EXAMPLE

```
C+
C This FORTRAN example program demonstrates the use of
C SMG$PUT_WITH_SCROLL.
C-
        PARAMETER       SMG$M_UP = 1
        PARAMETER       SMG$M_DOWN = 2
        PARAMETER       SMG$M_BOLD = 1
        PARAMETER       SMG$M_REVERSE = 2
        PARAMETER       SMG$M_BLINK = 4
        PARAMETER       SMG$M_UNDERLINE = 8
        IMPLICIT INTEGER*4 (A-Z)
C+
C Call SMG$CREATE_PASTEBOARD to establish the terminal screen
C as a pasteboard.
C-
        STATUS = SMG$CREATE_PASTEBOARD (NEW_PID)
        IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Using SMG$CREATE_VIRTUAL_DISPLAY, establish a virtual display region.
C-
```

```
      STATUS = SMG$CREATE_VIRTUAL_DISPLAY (5,80,DISPLAY_ID)
      IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Paste the virtual display to the screen, starting at
C row 10, column 15 by calling SMG$PASTE_VIRTUAL_DISPLAY.
C-
      STATUS = SMG$PASTE_VIRTUAL_DISPLAY(DISPLAY_ID,NEW_PID,10,15)
      IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Define a scrolling region through a call to
C SMG$SET_DISPLAY_SCROLL_REGION.
C-
      STATUS = SMG$SET_DISPLAY_SCROLL_REGION(DISPLAY_ID,1,5)
      IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Call SMG$PUT_WITH_SCROLL and SMG$ERASE_LINE to write three
C scrolling lines to the screen.  The first line will be underlined,
C the second blinking, and the third in reverse video.
C-
      DO I = 1,10
      IF ((I/2) + (I/2) .EQ. I) THEN
              DIR = SMG$M_UP
      ELSE
              DIR = SMG$M_DOWN
      ENDIF
      STATUS = SMG$PUT_WITH_SCROLL (DISPLAY_ID,
     1    'This line is  underlined',DIR,SMG$M_UNDERLINE,0)
      IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
      STATUS = SMG$ERASE_LINE(DISPLAY_ID)
      IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
      STATUS = SMG$PUT_WITH_SCROLL (DISPLAY_ID,'This line is blinking',
     1    DIR,SMG$M_BLINK,0)
      IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
      STATUS = SMG$ERASE_LINE (DISPLAY_ID)
      IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
      STATUS = SMG$PUT_WITH_SCROLL (DISPLAY_ID,'This line is reverse
     1    video',DIR,SMG$M_REVERSE,0)
      IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
      STATUS = SMG$ERASE_LINE (DISPLAY_ID)
      IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
      ENDDO
      END
```

This FORTRAN program calls Run-Time Library Screen Management routines to format screen output. When run, this program displays three scrolling lines which, respectively, are underlined, blinking, and in reverse video. Because it is not possible to represent scrolling (or blinking text) in a diagram, it is suggested that you run this example to observe the output.

SMG$READ_COMPOSED_LINE—Read Composed Line

SMG$READ_COMPOSED_LINE reads a line of input composed of normal keystrokes and equivalence strings.

FORMAT

SMG$READ_COMPOSED_LINE *keyboard-id ,key-table-id ,received-text [,prompt-string] [,received-string-length] [,display-id] [,function-keys-flag] [,ini-string] [,timeout] [,rendition-set] [,rendition-complement] [,terminator-code]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

keyboard-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the virtual keyboard from which input is to be read. The **keyboard-id** argument is the address of an unsigned longword that contains the keyboard identifier.

Keyboard-id is returned by SMG$CREATE_VIRTUAL_KEYBOARD.

key-table-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the key table to be used for translating keystrokes. The **key-table-id** argument is the address of an unsigned longword that contains the key table identifier.

Key-table-id is returned by SMG$CREATE_KEY_TABLE.

received-text

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

String into which SMG$READ_COMPOSED_LINE writes the complete composed line. The **received-text** argument is the address of a descriptor pointing to the storage in which the composed line is written.

prompt-string

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

String used to prompt for the read operation. The **prompt-string** argument is the address of a descriptor pointing to the prompt string.

received-string-length

VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

Receives the number of characters read or the maximum length of **received-text**, whichever is less. The **received-string-length** argument is the address of an unsigned longword into which SMG$READ_COMPOSED_LINE writes the number of characters read.

display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display identifier. The **display-id** argument is the address of an unsigned longword that contains the display identifier. This argument is optional only if you are not using the Screen Management Facility's output routines.

If you are using the Screen Management Facility input and output routines, this argument specifies the virtual display in which the input is to occur. The virtual display specified must be pasted to the same terminal as specified by **keyboard-id** and must not be occluded.

The input begins at the current virtual cursor position but the virtual cursor must be in column 1. Note that the length of the **prompt-string** plus the input is limited to the number of visible columns in the display.

Note: **This virtual display must be pasted in column 1 and may not have any other virtual displays to its right. This restriction is necessary because otherwise any occurrence of CTRL/R or CTRL/U would blank out the entire line, including any output pasted to the right. To circumvent this restriction, you may use SMG$REPAINT_LINE whenever a CTRL/R or CTRL/U is encountered.**

function-keys-flag

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Function keys. The **function-keys-flag** argument is the address of an unsigned longword. If **function-keys-flag** equals 1, then the function keys can be used and line editing is disabled. If **function-keys-flag** equals zero, line editing is enabled and the function keys cannot be used.

ini-string

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Optional string that contains the initial characters of the field. The **ini-string** argument is the address of a descriptor pointing to the string.

timeout

VMS Usage: **longword_signed**
type: **longword (signed)**
access: **read only**
mechanism: **by reference**

Optional timeout count. The **timeout** argument is the address of a signed longword containing the timeout count. If the **timeout** argument is specified, all characters entered before the timeout are returned in the buffer. If the **timeout** argument is omitted, characters are returned in the buffer until a terminator is encountered.

rendition-set

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Optional attribute specifier. The **rendition-set** argument is the address of a longword bit mask in which each 1-bit attribute causes the corresponding attribute to be set in the display. The following attributes can be specified by the **rendition-set** argument:

SMG$M_BLINK	Displays characters blinking.
SMG$M_BOLD	Displays characters in higher-than-normal intensity (bolded).
SMG$M_REVERSE	Displays characters in reverse video — that is, using the opposite default rendition of the virtual display.
SMG$M_UNDERLINE	Displays characters underlined.

The **display-id** argument must be specified when using the **rendition-set** argument.

rendition-complement

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Optional attribute complement specifier. The **rendition-complement** argument is the address of a longword bit mask **rendition-set** in which each 1-bit attribute causes the corresponding attribute to be complemented in the display. All of the attributes that can be specified with the **rendition-set** argument can be complemented with the **rendition-complement** argument. The **display-id** argument must be specified when the **rendition-complement** argument is used.

terminator-code

VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

Key terminator code. The **terminator-code** argument is an unsigned word into which is written a code indicating what character or key terminated the read. Key terminator codes are of the form SMG$K_TRM_keyname. The key names are listed in Table RTL–1, in Chapter 3.

DESCRIPTION

SMG$READ_COMPOSED_LINE reads a line composed of normal keystrokes and key equivalence strings as defined in the specified key table. Attributes of the key definition control whether the equivalence string is echoed and whether the read terminates with the defined keystroke. Normal keystrokes are always echoed.

The optional arguments **rendition-set** and **rendition-complement** let the user control the attributes of the virtual display in which the read is done. The **rendition-set** argument sets certain virtual display attributes, while **rendition-complement** complements these attributes. If the same bit is specified in both the **rendition-set** and **rendition-complement** parameters, the **rendition-set** is evaluated first, followed by the **rendition-complement**. By using these two parameters together, the user can control each virtual display attribute in a single procedure call. On a single attribute basis, the user can cause the following transformations:

Set	Complement	Action
0	0	Attribute unchanged.
1	0	Attribute set to "on."
0	1	Attribute set to complement of current setting.
1	1	Attribute set to "off."

A carriage return always terminates the read operation. If CTRL/Z is typed and there is no definition for CTRL/Z in the key definition table, "EXIT" is echoed and the read is terminated. If CTRL/Z was the first character typed on the line, SMG$_EOF is returned. Otherwise, SMG$_EOF is returned on the next read operation. SMG$_EOF is also returned if RMS is used for the input operation and it returns RMS$_EOF. No other terminators are recognized except those specified as attributes in a key definition.

If the arrow keys and CTRL/B are not defined, the previous lines read with the SMG$READ_xxxx routines can be recalled using the arrow keys. The number of lines saved for later recall depends upon the **recall-size** argument in SMG$CREATE_VIRTUAL_KEYBOARD. The default is 20 lines.

Note that SMG$READ_COMPOSED_LINE calls the SMG$FLUSH_BUFFER routine before performing the input operation. This ensures that the screen image is up to date at the time of the input operation. Display batching for both the pasteboard and virtual display must be off when you use SMG$READ_COMPOSED_LINE.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SS$_CANCEL	I/O operation canceled while queued (by SMG$CANCEL_INPUT).
SS$_ABORT	I/O operation aborted during execution (by SMG$CANCEL_INPUT).
SMG$_EOF	End of file.
SMG$_INVDIS_ID	Invalid **display-id**.
SMG$_INVKBD_ID	Invalid **keyboard-id**.
SMG$_INVKTB_ID	Invalid **key-table-id**.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_ILLBATFNC	Input not allowed from a batched display.
SMG$_INVCOL	Invalid column. The read operation attempts to use a column outside the virtual display.

Any condition values returned by LIB$COPY_R_DX.

Any condition values returned by $GET (except RMS$_EOF).

Any condition values returned by $QIOW.

SMG$READ_FROM_DISPLAY—Read Text from Display

SMG$READ_FROM_DISPLAY reads a line of text from a virtual display.

FORMAT

SMG$READ_FROM_DISPLAY *display-id ,returned-string [,terminator-string][,row]*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

display-id

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Specifies the virtual display from which text is read. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

Display-id is returned by SMG$CREATE_VIRTUAL_DISPLAY.

returned-string

VMS Usage:	**char_string**
type:	**character string**
access:	**write only**
mechanism:	**by descriptor**

String into which SMG$READ_FROM_DISPLAY writes the information read from the virtual display. The **returned-string** argument is the address of a descriptor pointing to the storage into which the string is written.

terminator-string

VMS Usage:	**char_string**
type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

String containing a terminator or terminators that end the backward search, thus determining the starting position of the returned string. The **terminator-string** argument is the address of a descriptor pointing to the string of terminators. If omitted, no back searching is performed; the returned string starts with the character at the current cursor position.

row

VMS Usage: **longword_signed**
type: **longword (signed)**
access: **read only**
mechanism: **by reference**

The **row** argument is the address of a signed longword that contains the row of the **display-id** to read from. This is an optional argument.

DESCRIPTION

SMG$READ_FROM_DISPLAY returns a string that contains some or all of the text on the current line of the specified virtual display. If the **terminator-string** argument is omitted, the contents of the current line (from the current column position to the rightmost column position) are returned. If the **row** argument is passed, the contents of line **row** from column 1 to the rightmost column is returned in **returned-string**. If the **row** argument is passed, the **terminator-string** argument is ignored.

If you specify a **terminator-string**, each character in it serves as a terminator for "back searching," that is, the process of determining the first character position to be returned. If none of the specified terminators are encountered, the search is terminated at the first character position on the line.

Note that SMG$READ_FROM_DISPLAY calls the SMG$FLUSH_BUFFER routine before performing the input operation. This ensures that the screen image is up to date at the time of the read operation.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVDIS_ID	Invalid **display-id**.
LIB$_INVSTRDES	Invalid string descriptor.
LIB$_INSVIRMEM	Insufficient virtual memory.

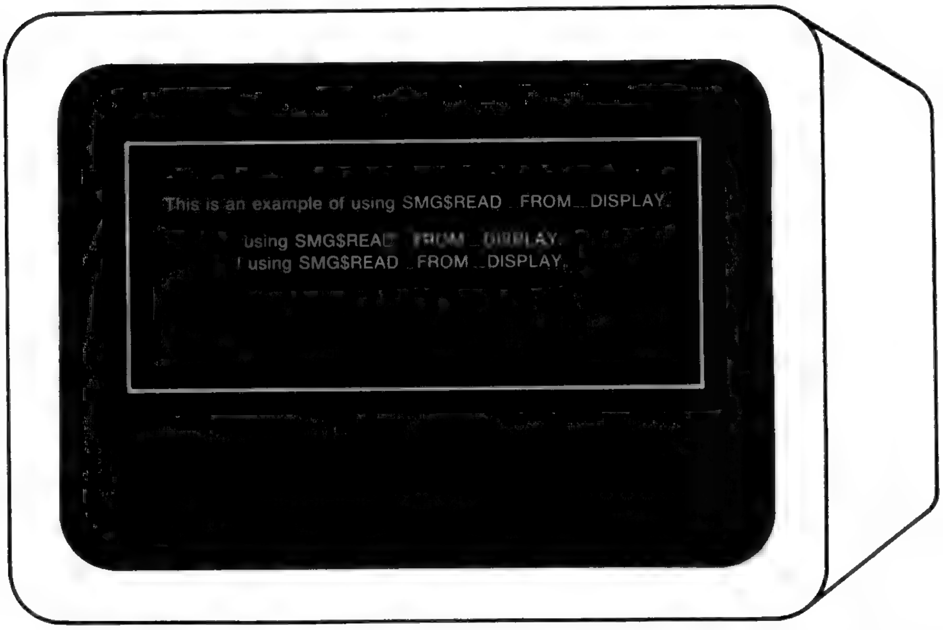
EXAMPLE

```
C+
C This FORTRAN example demonstrates the use of SMG$READ_FROM_DISPLAY.
C-
C+
C Include the SMG definitions. In particular, we want SMG$M_BORDER.
C-
        IMPLICIT INTEGER (A-Z)
        INCLUDE '($SMGDEF)'
        CHARACTER*80 TEXT
C+
C Use SMG$CREATE_VIRTUAL_DISPLAY to create the virtual display
C and give it a border.
C-
        ROWS = 5
        COLUMNS = 60
        STATUS = SMG$CREATE_VIRTUAL_DISPLAY
     1  (ROWS, COLUMNS, DISPLAY1, SMG$M_BORDER)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Create the pasteboard by calling SMG$CREATE_PASTEBOARD.
C-
        STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
```

```
C+
C Call SMG$PASTE_VIRTUAL_DISPLAY and SMG$PUT_LINE to paste
C the virtual display and put some text on line 2.
C-
        STATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DISPLAY1, PASTE1, 2, 10)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        STATUS = SMG$PUT_LINE (DISPLAY1, ' ')
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        STATUS = SMG$PUT_LINE (DISPLAY1,
     1          'This is an example of using SMG$READ_FROM_DISPLAY.')
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Use SMG$READ_FROM_DISPLAY to read line 2 from the virtual
C display, starting at column 22.
C-
        STATUS = SMG$SET_CURSOR_ABS ( DISPLAY1, 2, 22)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Search line 2 from column 22 to column 1 for the null string.
C Since no terminator will be supplied, no "back-searching" will take
C place. TEXT will be assigned the "value" of the line from
C column 22 to the rightmost column.
C-
        STATUS = SMG$READ_FROM_DISPLAY ( DISPLAY1, TEXT)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Put the line of text found into the virtual display at row 4,
C column 10 by calling SMG$SET_CURSOR_ABS and SMG$PUT_LINE.
C-
        STATUS = SMG$SET_CURSOR_ABS ( DISPLAY1, 4, 10)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        STATUS = SMG$PUT_LINE (DISPLAY1, TEXT)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Use SMG$SET_CURSOR_ABS to set the cursor back to line 2, column 22.
C-
        STATUS = SMG$SET_CURSOR_ABS ( DISPLAY1, 2, 22)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Use SMG$READ_FROM_DISPLAY to search line 2 from column 22 to
C column 1 for an "f".  Now, "back-searching" will take place.
C Starting at column 22, "back-track" to column 1 looking for "f".
C Text will then be assigned the "value" of the line from the
C present cursor position (where the "f" is, to the rightmost
C column.
C-
        STATUS = SMG$READ_FROM_DISPLAY ( DISPLAY1, TEXT, 'f')
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Put the line of text found into the virtual display at row 4, column 10.
C-
        STATUS = SMG$SET_CURSOR_ABS ( DISPLAY1, 5, 10)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        STATUS = SMG$PUT_LINE (DISPLAY1, TEXT)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        END
```

The output generated by this FORTRAN program is shown in Figure RTL-32.

Figure RTL-32 Output Generated by FORTRAN Program Calling SMG$READ_FROM_DISPLAY



ZK-4134-85

SMG$READ_KEYSTROKE—Read a Single Character

SMG$READ_KEYSTROKE reads a keystroke and returns that keystroke's terminator code.

FORMAT

SMG$READ_KEYSTROKE *keyboard-id ,terminator-code [,prompt-string] [,timeout] [,display-id] [,rendition-set] [,rendition-complement]*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

keyboard-id

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Keyboard identifier. The **keyboard-id** argument is an unsigned longword containing the identification of the virtual keyboard from which to read. A virtual keyboard is created by calling the SMG$CREATE_VIRTUAL_KEYBOARD routine.

terminator-code

VMS Usage:	**word_unsigned**
type:	**word (unsigned)**
access:	**write only**
mechanism:	**by reference**

Key terminator code. The **terminator-code** argument is an unsigned word into which is written a code indicating what character or key terminated the read. Key terminator codes are of the form SMG$K_TRM_keyname. The key names are listed in Table RTL–1, (Section 3.5.1) in Part I, Chapter 3 of this manual.

prompt-string

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Prompt string. The **prompt-string** argument is an optional string that is used as the prompt for the read operation.

timeout

VMS Usage: **longword_signed**
type: **longword (signed)**
access: **read only**
mechanism: **by reference**

Timeout count. The **timeout** argument is optional. If specified, any character typed before the timeout is returned in the buffer.

display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display identifier. The optional **display-id** argument is the address of an unsigned longword that contains the identifier of the virtual display in which the read is to be performed. If the optional **prompt-string** argument is specified while there are multiple virtual displays pasted, the **display-id** argument is required to determine in which virtual display the prompt string will be written. If the **prompt-string** argument is not specified, then do not specify the **display-id** argument.

In the case of multiple virtual displays, each virtual display has an associated virtual cursor position. At the same time, there is a single physical cursor position corresponding to the current location of the physical cursor. If the **display-id** argument is specified, the read begins at the current virtual cursor position in the specified virtual display. If omitted, the read begins in the current physical cursor position. Note that the length of the **prompt-string** plus the key entered is limited to the number of visible columns in the display.

rendition-set

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute specifier. The optional **rendition-set** argument is the address of a longword bit mask in which each 1-bit attribute causes the corresponding attribute to be set in the display. The following attributes can be specified by the **rendition-set** argument:

SMG$M_BLINK	Displays characters blinking.
SMG$M_BOLD	Displays characters in higher-than-normal intensity (bolded).
SMG$M_REVERSE	Displays characters in reverse video — that is, using the opposite default rendition of the virtual display.
SMG$M_UNDERLINE	Displays characters underlined.

The **display-id** argument must be specified when using the **rendition-set** argument.

rendition-complement

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute complement specifier. The optional **rendition-complement** argument is the address of a longword bit mask in which each 1-bit attribute causes the corresponding attribute to be complemented in the display. All of the attributes that can be specified with the **rendition-set** argument can be complemented with the **rendition-complement** argument. The **display-id** argument must be specified when using the **rendition-complement** argument.

DESCRIPTION

SMG$READ_KEYSTROKE reads a keystroke from the virtual keyboard specified and returns the terminator code of that keystroke in the form SMG$K_TRM_keyname. The keystroke entered to be read is not echoed on the screen. This keystroke may be any standard alphabetic character, any keypad or function key, or one of the directional arrows.

The optional arguments **rendition-set** and **rendition-complement** let the user control the attributes of the virtual display in which the **prompt-string** is printed. The **rendition-set** argument sets certain virtual display attributes, while **rendition-complement** complements these attributes. If the same bit is specified in both the **rendition-set** and **rendition-complement** parameters, the **rendition-set** is evaluated first, followed by the **rendition-complement**. By using these two parameters together, the user can control each virtual display attribute in a single procedure call. On a single-attribute basis, the user can cause the following transformations:

Set	Complement	Action
0	0	Attribute unchanged.
1	0	Attribute set to "on."
0	1	Attribute set to complement of current setting.
1	1	Attribute set to "off."

Note that display batching for both the pasteboard and the virtual display must be off when you use SMG$READ_KEYSTROKE.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SS$_CANCEL	I/O operation canceled while queued (by SMG$CANCEL_INPUT).
SS$_ABORT	I/O operation aborted during execution (by SMG$CANCEL_INPUT).
SMG$_EOF	End-of-file.
SMG$_INVDIS_ID	Invalid display-id.
SMG$_INVKBD_ID	Invalid keyboard-id.
SMG$_WRONUMARG	Wrong number of arguments.
LIB$_xxx	Any error from LIB$SCOPY_R_DX.
RMS$_xxx	Any error from $GET (except RMS$_EOF).
SS$_xxx	Any error from $QIOW.

EXAMPLES

1

```
C+
C This FORTRAN example program demonstrates the use of
C SMG$READ_KEYSTROKE.
C-
C+
C This routine creates a virtual display and writes it to the pasteboard.
C Data is placed in the virtual display via SMG$PUT_CHARS.
C
C First, include the SMG definitions. In particular, we want SMG$M_BORDER.
C-
        IMPLICIT INTEGER (A-Z)
        INCLUDE '($SMGDEF)'
        CHARACTER*3 TEXT
        CHARACTER*27 TEXT_OUTPUT
C+
C Use SMG$CREATE_VIRTUAL_DISPLAY to create a virtual
C display with a border.
C-
        ROWS = 7
        COLUMNS = 60
        STATUS = SMG$CREATE_VIRTUAL_DISPLAY
     1          (ROWS, COLUMNS, DISPLAY1, SMG$M_BORDER)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Create the pasteboard using SMG$CREATE_PASTEBOARD.
C-
        STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Use SMG$CREATE_VIRTUAL_KEYBOARD to create a virtual keyboard.
C-
        STATUS = SMG$CREATE_VIRTUAL_KEYBOARD ( KEYBOARD1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Using SMG$PASTE_VIRTUAL_DISPLAY, paste the virtual display
C at row 3, column 9.
C-
        STATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DISPLAY1, PASTE1, 3, 9)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        STATUS = SMG$PUT_LINE (DISPLAY1,
     1          'Enter the character K after the >> prompt.')
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
```

```
        STATUS = SMG$PUT_LINE (DISPLAY1,
     1          'This character will not be echoed as you type it.')
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        STATUS = SMG$PUT_LINE (DISPLAY1,
     1  'The terminal character equivalent of K is displayed.')
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        STATUS = SMG$PUT_LINE (DISPLAY1, ' ')
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Call SMG$READ_KEYSTROKE to read a keystroke from the virtual
C pasteboard.
C-
        STATUS = SMG$READ_KEYSTROKE ( KEYBOARD1, TERM_CHAR, '>>', ,
     1          DISPLAY1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        STATUS = SMG$PUT_LINE (DISPLAY1, ' ')
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Use OTS$CVT_L_TI to convert the decimal value of TERM_CHAR to
C a decimal ASCII text string.
C-
        STATUS = OTS$CVT_L_TI( TERM_CHAR, TEXT)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        TEXT_OUTPUT = ' TERMINAL CHARACTER IS: ' // TEXT
C+
C Call SMG$PUT_LINE and SMG$PUT_CHARS to print the decimal
C ASCII text string.
C-
        STATUS = SMG$PUT_LINE (DISPLAY1, TEXT_OUTPUT)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        STATUS = SMG$PUT_CHARS (DISPLAY1, TEXT, 7, 25)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        END
```

The output generated by this FORTRAN program is shown in Figure RTL-33.

Figure RTL-33 Output Generated by FORTRAN Program Calling SMG$READ_KEYSTROKE

ZK-4129-85

2

```
1       OPTION TYPE=EXPLICIT
        !+
        ! This routine demonstrates the use of SMG$READ_KEYSTROKE to read
        ! a keystroke from the terminal.
        !
        ! Build this program using the following commands.
        !
        !$ BASIC READ_KEY
        !$ CREATE SMGDEF.MAR
        !       .TITLE  SMGDEF - Define SMG$ constants
        !       .Ident  /1-000/
        !
        !       $SMGDEF GLOBAL
        !
        !       .END
        !$ MACRO SMGDEF
        !$ LINK READ_KEY,SMGDEF
        !
        !-
        DECLARE LONG kb_id, ret_status, term_code, I, timer
        EXTERNAL SUB LIB$SIGNAL( LONG BY VALUE )
        EXTERNAL SUB LIB$STOP( LONG BY VALUE )
        EXTERNAL LONG CONSTANT SS$_TIMEOUT
        EXTERNAL LONG CONSTANT SMG$K_TRM_PF1
        EXTERNAL LONG CONSTANT SMG$K_TRM_PERIOD
        EXTERNAL LONG CONSTANT SMG$K_TRM_UP
        EXTERNAL LONG CONSTANT SMG$K_TRM_RIGHT
        EXTERNAL LONG CONSTANT SMG$K_TRM_F6
        EXTERNAL LONG CONSTANT SMG$K_TRM_F20
        EXTERNAL LONG CONSTANT SMG$K_TRM_FIND
        EXTERNAL LONG CONSTANT SMG$K_TRM_NEXT_SCREEN
        EXTERNAL LONG CONSTANT SMG$K_TRM_TIMEOUT
        EXTERNAL LONG FUNCTION SMG$CREATE_VIRTUAL_KEYBOARD( LONG, STRING )
        EXTERNAL LONG FUNCTION SMG$DELETE_VIRTUAL_KEYBOARD( LONG )
        EXTERNAL LONG FUNCTION SMG$READ_KEYSTROKE( LONG, LONG, STRING, &
```

```
    LONG, LONG )

!+
! Prompt the user for the timer value.  A value of 0 will cause
! the type ahead buffer to be read.
!-
INPUT "Enter timer value (0 to read typeahead buffer):  ";timer
!+
! Establish a SMG connection to SYS$INPUT.  Signal any unexpected
! errors.
!-
ret_status = SMG$CREATE_VIRTUAL_KEYBOARD( kb_id, "SYS$INPUT:" )
IF (ret_status AND 1%) = 0% THEN
    CALL LIB$SIGNAL( ret_status )
END IF
!+
!   Read a keystoke, tell the user what we found.
!-
ret_status = SMG$READ_KEYSTROKE( kb_id, term_code, , timer, )
IF (ret_status <> SS$_TIMEOUT) AND ((ret_status AND 1%) = 0%) THEN
    CALL LIB$SIGNAL( ret_status )
END IF
PRINT "term_code = ";term_code
SELECT term_code
    CASE 0 TO 31
        PRINT "You typed a control character"
    CASE 32 TO 127
        PRINT "You typed: ";CHR$(term_code)
    CASE SMG$K_TRM_PF1 TO SMG$K_TRM_PERIOD
        PRINT "You typed one of the keypad keys"
    CASE SMG$K_TRM_UP TO SMG$K_TRM_RIGHT
        PRINT "You typed one of the cursor positioning keys"
    CASE SMG$K_TRM_F6 TO SMG$K_TRM_F20
        PRINT "You typed one of the function keys"
    CASE SMG$K_TRM_FIND TO SMG$K_TRM_NEXT_SCREEN
        PRINT "You typed one of the editing keys"
    CASE SMG$K_TRM_TIMEOUT
        PRINT "You did not type a key fast enough"
    CASE ELSE
        PRINT "I'm not sure what key you typed"
END SELECT
!+
! Close the connection to SYS$INPUT, and signal any errors.
!-
ret_status = SMG$DELETE_VIRTUAL_KEYBOARD( kb_id )
IF (ret_status AND 1%) = 0% THEN
    CALL LIB$SIGNAL( ret_status )
END IF
END
```

This BASIC program reads a keystroke and returns the term_code and the name of the keystroke entered. One sample of the output generated by this program is as follows:

```
$ BASIC READ_KEY
$ MACRO SMGDEF
$ LINK READ_KEY,SMGDEF
$ RUN READ_KEY
Enter the timer value (0 to read typeahead buffer): ?  9
term_code = 100
You typed: d
```

Note that in this example, the user entered the keystroke "d" following the first prompt. The keystroke entered was not echoed.

SMG$READ_STRING—Read String

SMG$READ_STRING reads a string from a virtual keyboard.

FORMAT

SMG$READ_STRING *keyboard-id ,received-text*
[,prompt-string]
[,max-length]
[,modifiers] [,timeout]
[,terminator-set]
[,received-string-length]
[,terminator-code]
[,display-id]
[,ini-string] [,rendition-set]
[,rendition-complement]

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

keyboard-id

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Specifies the virtual keyboard from which input is to be read. The **keyboard-id** argument is the address of an unsigned longword that contains the keyboard identifier.

Keyboard-id is returned by SMG$CREATE_VIRTUAL_KEYBOARD.

received-text

VMS Usage:	**char_string**
type:	**character string**
access:	**write only**
mechanism:	**by descriptor**

String into which the input line is written. The **received-text** argument is the address of a descriptor pointing to the storage into which the text is written.

prompt-string

VMS Usage:	**char_string**
type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

String used to prompt for the read operation. The **prompt** argument is the address of a descriptor pointing to the prompt string.

max-length

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the maximum number of characters to be read. The **max-length** argument is the address of a signed longword integer that contains the maximum number of characters to be read. The maximum valid value for this argument is 512. If omitted, 512 is the default.

modifiers

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Bit mask that specifies optional behavior. The **modifiers** argument is the address of an unsigned longword that contains the bit mask.

Valid modifiers are:

- TRM$M_TM_CVTLOW
- TRM$M_TM_NOECHO
- TRM$M_TM_PURGE
- TRM$M_TM_TRMNOECHO
- TRM$M_TM_NOEDIT
- TRM$M_TM_NORECALL

See the terminal driver section of the *VAX/VMS I/O User's Reference Manual: Part I* for more information on modifiers. The TRM$ symbols are defined by the $TRMDEF macro/module in DIGITAL-supplied system symbol libraries.

timeout

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the number of seconds allowed between the time the prompt is issued and the completion of the input operation. The **timeout** argument is the address of a signed longword integer that contains the number of seconds.

If **timeout** is specified, all characters typed before the expiration time are returned in **received-text**. If omitted, the input operation remains active until a terminator is typed.

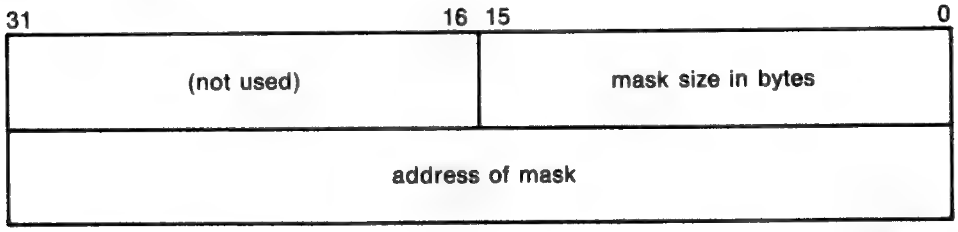
terminator-set

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor, fixed length**

Either a mask that specifies which characters are to be treated as terminators (short form) or a descriptor pointing to such a mask (long form). The **terminator-set** argument is the address of a descriptor pointing to the mask.

If you want to use terminators with ASCII values in the range 0 to 31, use the short form: You create this mask by setting the bit that corresponds to the ASCII value of desired terminator. For example, to specify that CTRL/A (ASCII value 1) is a terminator, you set bit 1 in the **terminator-set** mask.

If you want to use terminators with ASCII values outside the range 0 to 31, use the long form; you first create a descriptor of this form:



ZK-2004-84

The mask itself has the same format as that of the short form; however, the long form allows use of a more comprehensive set of terminator characters. For example, a mask size of 16 bytes allows any 7-bit ASCII character to be set as a terminator, while a mask size of 32 bytes allows any 8-bit character to be set as a terminator. Any mask size between 1 and 32 bytes is acceptable.

If the terminator mask is all zeros, there are no specified terminators and the read terminates when the number of characters specified in the **max-length** argument have been transferred.

If the **terminator-set** argument is omitted, the set of terminators is the VMS default terminator set. For more information see the *VAX/VMS I/O User's Reference Manual: Part I*, Section 8.4.1.2.

received-string-length

VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

Receives the number of characters read or the maximum size of **received-text**, whichever is less. The **received-string-length** argument is the address of an unsigned word into which is written the number of characters or the maximum size.

terminator-code

VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

Key terminator code. The **terminator-code** argument is an unsigned word into which is written a code indicating what character or key terminated the read. Key terminator codes are of the form SMG$K_TRM_keyname. The keynames are listed in Table RTL–1, in Chapter 3.

display-id
VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display identifier. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

This argument is optional only if you are not using the Screen Management Facility's output routines.

If you are using the Screen Management Facility input and output routines, this argument specifies the virtual display in which the input is to occur. The virtual display specified must be pasted to the same terminal as specified by **keyboard-id** and must not be occluded.

This virtual display must be pasted in column 1 and may not have any other virtual displays to its right. This restriction applies because otherwise the occurrence of a CTRL/R or CTRL/U would cause the entire line to be blanked, including any output to the right. To circumvent this restriction, you may use SMG$REPAINT_LINE to repaint the line when a CTRL/R or CTRL/U is detected.

The input begins at the current cursor position but the cursor must be in column 1. Note that the length of the prompt plus the input is limited to the number of visible columns in the display.

ini-string
VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Initial character string. The **ini-string** argument is the address of a descriptor pointing to the optional string that contains the initial characters of the field.

rendition-set
VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute specifier. The optional **rendition-set** argument is the address of a longword bit mask in which each 1-bit attribute causes the corresponding attribute to be set in the display. The following attributes can be specified using the **rendition-set** argument:

SMG$M_BLINK	Displays characters blinking.
SMG$M_BOLD	Displays characters in higher-than-normal intensity (bolded).
SMG$M_REVERSE	Displays characters in reverse video — that is, using the opposite default rendition of the virtual display.
SMG$M_UNDERLINE	Displays characters underlined.

The **display-id** argument must be specified when using the **rendition-set** argument.

rendition-complement

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute complement specifier. The optional **rendition-complement** argument is the address of a longword bit mask in which each 1-bit attribute causes the corresponding attribute to be complemented in the display. All of the attributes that can be specified with the **rendition-set** argument can be complemented with **rendition-complement**. The **display-id** argument must be specified when using the **rendition-complement** argument.

DESCRIPTION

SMG$READ_STRING returns a string of characters read from a virtual display. Note that display batching for both the pasteboard and the virtual display must be off when you use SMG$READ_STRING.

The optional arguments **rendition-set** and **rendition-complement** let the user control the attributes of the virtual display in which the read is done. The **rendition-set** argument sets certain virtual display attributes, while **rendition-complement** complements these attributes. If the same bit is specified in both the **rendition-set** and **rendition-complement** parameters, the **rendition-set** is evaluated first, followed by **rendition-complement**. By using these two parameters together, the user can control each virtual display attributes in a single procedure call. On a single-attribute basis, the user can cause the following transformations:

Set	Complement	Action
0	0	Attribute unchanged.
1	0	Attribute set to "on."
0	1	Attribute set to complement of current setting.
1	1	Attribute set to "off."

Note that the text read by SMG$READ_STRING is saved for later recall with SMG$READ_COMPOSED_LINE.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SS$_CANCEL	I/O operation canceled while queued (by SMG$CANCEL_INPUT).
SS$_ABORT	I/O operation aborted during execution (by SMG$CANCEL_INPUT).
SMG$_EOF	End of file.
SMG$_INVDIS_ID	Invalid **display-id**.
SMG$_INVKBD_ID	Invalid **keyboard-id**.
SMG$_INVKTB_ID	Invalid **key-table-id**.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_ILLBATFNC	Input not allowed from a batched display.
SMG$_INVCOL	Invalid column. The input occurs outside the virtual display.

SMG$_INVMAXLEN	Maximum length specified was greater than 512.

Any condition values returned by LIB$COPY_R_DX.

Any condition values returned by $GET (except RMS$_EOF).

Any condition values returned by $QIOW.

EXAMPLES

1

```
1       OPTION TYPE=EXPLICIT
        !+
        ! This routine demonstrates the use of SMG$READ_STRING to read
        ! either a string, a control key, or a keypad key.
        !
        ! Build this program using the following commands.
        !
        !$ BASIC SMGTEST
        !$ CREATE SMGDEF.MAR
        !       .TITLE  SMGDEF - Define SMG$ constants
        !       .Ident  /1-000/
        !
        !       $SMGDEF GLOBAL
        !
        !       .END
        !$ MACRO SMGDEF
        !$ LINK SMGTEST,SMGDEF
        !
        !-
        DECLARE LONG KB_ID, RET_STATUS, STR_LEN, TERM_CODE, MODIFIER, I, TIMER
        DECLARE STRING DATA_STR, TERM_SET
        EXTERNAL LONG CONSTANT IO$M_TIMED
        EXTERNAL LONG CONSTANT IO$M_NOECHO
        EXTERNAL LONG CONSTANT IO$M_NOFILTR
        EXTERNAL SUB LIB$SIGNAL( LONG BY VALUE )
        EXTERNAL SUB LIB$STOP( LONG BY VALUE )
        EXTERNAL LONG CONSTANT SS$_TIMEOUT
        EXTERNAL LONG CONSTANT SMG$K_TRM_PF1
        EXTERNAL LONG CONSTANT SMG$K_TRM_PERIOD
        EXTERNAL LONG CONSTANT SMG$K_TRM_UP
        EXTERNAL LONG CONSTANT SMG$K_TRM_RIGHT
        EXTERNAL LONG CONSTANT SMG$K_TRM_F6
        EXTERNAL LONG CONSTANT SMG$K_TRM_F20
        EXTERNAL LONG CONSTANT SMG$K_TRM_E1
        EXTERNAL LONG CONSTANT SMG$K_TRM_E6
        EXTERNAL LONG CONSTANT SMG$K_TRM_TIMEOUT
        EXTERNAL LONG FUNCTION SMG$CREATE_VIRTUAL_KEYBOARD( LONG, STRING )
        EXTERNAL LONG FUNCTION SMG$DELETE_VIRTUAL_KEYBOARD( LONG )
        EXTERNAL LONG FUNCTION SMG$READ_STRING( LONG, STRING, STRING, &
            LONG, LONG, LONG, STRING, LONG, LONG )

        !+
        ! Prompt the user for the timer value.  A value of 0 will cause
        ! the type ahead buffer to be read.
        !-
        INPUT "Enter timer value (0 to read typeahead buffer):  ";TIMER
        !+
        ! Tell SMG to use the timer value
        !-
        MODIFIER = IO$M_TIMED
        !+
        ! Establish a SMG connection to SYS$INPUT.  Signal any unexpected
        ! errors.
        !-
        RET_STATUS = SMG$CREATE_VIRTUAL_KEYBOARD( KB_ID, "SYS$INPUT:" )
```

```
IF (RET_STATUS AND 1%) = 0% THEN
    CALL LIB$SIGNAL( RET_STATUS )
END IF

!+
! Tell SMG to use any keystroke except a letter or number
! as a terminator to the input and perform the read.
! Signal any error except SS$_TIMEOUT
!-
TERM_SET = STRING$( 4%, -1% ) + STRING$(12%, 0%)
RET_STATUS = SMG$READ_STRING( KB_ID, DATA_STR, , ,  &
        MODIFIER, TIMER, TERM_SET, &
        STR_LEN, TERM_CODE )
IF (RET_STATUS <> SS$_TIMEOUT) AND ((RET_STATUS AND 1%) = 0%) THEN
    CALL LIB$SIGNAL( RET_STATUS )
END IF

!+
! All the data should come back as a terminator code, since any
! character can be a terminator.
!-
PRINT "data string = ";LEFT(DATA_STR, STR_LEN)
PRINT "term_code = ";TERM_CODE
SELECT TERM_CODE
    CASE 0 TO 31
        PRINT "You typed a control character"
    CASE 32 TO 127
        PRINT "You typed: ";CHR$(TERM_CODE)
    CASE SMG$K_TRM_PF1 TO SMG$K_TRM_PERIOD
        PRINT "You typed one of the keypad keys"
    CASE SMG$K_TRM_UP TO SMG$K_TRM_RIGHT
        PRINT "You typed one of the cursor positioning keys"
    CASE SMG$K_TRM_F6 TO SMG$K_TRM_F20
        PRINT "You typed one of the function keys"
    CASE SMG$K_TRM_E1 TO SMG$K_TRM_E6
        PRINT "You typed one of the editing keys"
    CASE SMG$K_TRM_TIMEOUT
        PRINT "You did not type a key fast enough"
    CASE ELSE
        PRINT "I'm not sure what key you typed"
END SELECT

!+
! Close the connection to SYS$INPUT, and signal any errors.
!-
RET_STATUS = SMG$DELETE_VIRTUAL_KEYBOARD( KB_ID )
IF (RET_STATUS AND 1%) = 0% THEN
    CALL LIB$SIGNAL( RET_STATUS )
END IF

END
```

This BASIC example program demonstrates the use of SMG$READ_STRING. One sample of the output generated by this program is as follows:

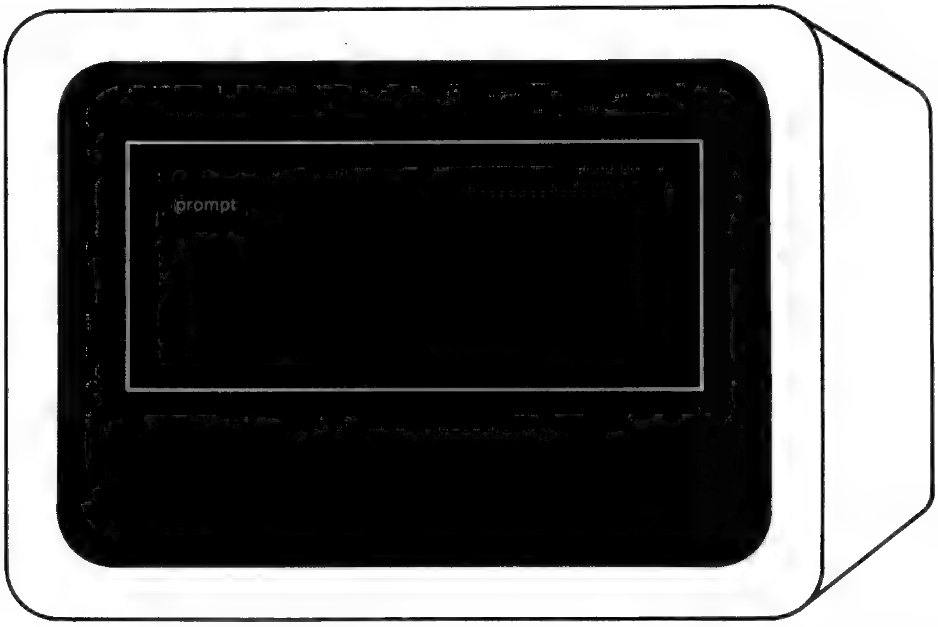
```
Enter timer value (0 to read typeahead buffer): ?  5
 d
data string = d
term_code = 13
You typed a control character
```

2

```
C+
C This FORTRAN example program demonstrates how to use
C SMG$READ_STRING.
C
C This routine creates a virtual display and writes it to the pasteboard.
C Data is placed in the virtual display via SMG$PUT_CHARS.
C-
C+
C Include the SMG definitions. In particular, we want SMG$M_BORDER.
C-
        IMPLICIT INTEGER (A-Z)
        INCLUDE '($SMGDEF)'
        CHARACTER*20 TEXT
C+
C Create a virtual display with a border using SMG$CREATE_VIRTUAL_DISPLAY.
C-
        ROWS = 7
        COLUMNS = 50
        STATUS = SMG$CREATE_VIRTUAL_DISPLAY
     1          (ROWS, COLUMNS, DISPLAY1, SMG$M_BORDER)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Use SMG$CREATE_PASTEBOARD to create the pasteboard.
C-
        STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Create a virtual keyboard by calling SMG$CREATE_VIRTUAL_KEYBOARD.
C-
        STATUS = SMG$CREATE_VIRTUAL_KEYBOARD ( KEYBOARD1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Use SMG$PASTE_VIRTUAL_DISPLAY to paste the virtual display
C at row 3, column 9.
C-
        STATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DISPLAY1, PASTE1, 3, 9)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Read a string from the virtual pasteboard using SMG$READ_STRING.
C-
        STATUS = SMG$READ_STRING ( KEYBOARD1,
     1          TEXT, 'prompt', 20, , , , , , DISPLAY1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        END
```

The output generated by this FORTRAN program before the call to SMG$READ_STRING is shown in Figure RTL–34. The program is waiting for input. The cursor immediately follows the word prompt.

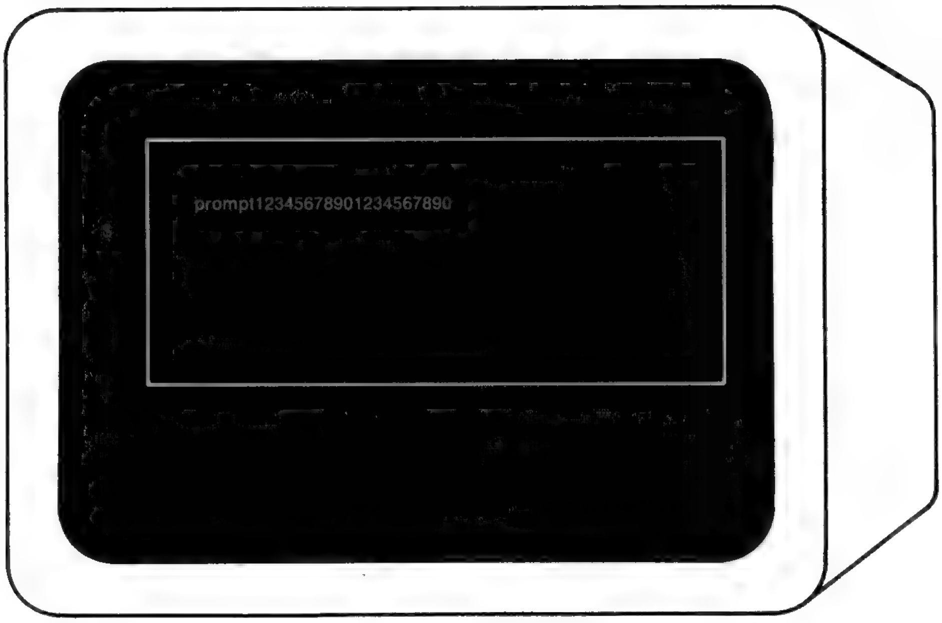
Figure RTL-34 Output Generated Before the Call to SMG$READ_STRING



ZK-4138-85

The output generated after the call to SMG$READ_STRING is shown in Figure RTL-35.

Figure RTL-35 Output Generated After the Call to SMG$READ_STRING



ZK-4140-85

SMG$READ_VERIFY—Read and Verify a String

SMG$READ_VERIFY reads a sequence of characters and verifies the sequence.

FORMAT

SMG$READ_VERIFY
keyboard-id ,out-string
,in-string ,pic-string
,fill-char ,clear-char
[,prompt-string] [,modifiers]
[,timeout] [,terminator-set]
[,ini-offset] [,terminator-code]
[,display-id] [,alt-echo-string]
[,alt-display-id] [,rendition-set]
[,rendition-complement]

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

keyboard-id

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Keyboard identifier. The **keyboard-id** argument is the address of an unsigned longword integer containing the identifier of the virtual keyboard from which to read. The virtual keyboard is created by calling the SMG$CREATE_VIRTUAL_KEYBOARD routine.

out-string

VMS Usage:	**char_string**
type:	**character string**
access:	**write only**
mechanism:	**by descriptor**

Output string into which SMG$READ_VERIFY writes the characters that are read. The **out-string** argument is the address of a descriptor pointing to this output string.

in-string

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

Input string that contains the initial characters of the field. The **in-string** argument is the address of a descriptor pointing to the input string.

pic-string

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Picture string that contains a picture of what the field is to look like. The **pic-string** argument is the address of a descriptor pointing to the picture string.

For more information on the legal values for the **pic-string** argument, see Section 8.4.1.4 in the *VAX/VMS I/O User's Reference Manual: Part I.*

fill-char

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Fill character. The **fill-char** argument is the address of a descriptor pointing to the string that contains the character to be used as a fill character in the **in-string** argument.

For more information on the **fill-char** parameter, see Section 8.4.1.4 in the *VAX/VMS I/O User's Reference Manual: Part I.*

clear-char

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Clear character. The **clear-char** argument is the address of a descriptor pointing to the string that contains the character to be displayed for each occurrence of **fill-char** in **in-string**.

For more information on the **clear-char** argument, see Section 8.4.1.4 in the *VAX/VMS I/O User's Reference Manual: Part I.*

prompt-string

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Prompt string. The **prompt-string** argument is the address of a descriptor pointing to the string that SMG$READ_VERIFY uses as the prompt for the read operation. This is an optional argument.

modifiers

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Modifiers. The **modifiers** argument is a longword bit mask that specifies optional behavior. The bits defined are the same as for the $QIO item-list entry TRM$_MODIFIERS. This is an optional argument.

Valid modifiers are:

- TRM$M_TM_CVTLOW
- TRM$M_TM_NOECHO
- TRM$M_TM_PURGE
- TRM$M_TM_TRMNOECHO
- TRM$M_TM_NOEDIT
- TRM$M_TM_NORECALL

See the terminal driver section of the *VAX/VMS I/O User's Reference Manual: Part I* for more information on modifiers. The TRM$ symbols are defined by the $TRMDEF macro/module in DIGITAL-supplied system symbol libraries.

timeout

VMS Usage: **longword_signed**
type: **longword (signed)**
access: **read only**
mechanism: **by reference**

Timeout count. The **timeout** argument is optional. If specified, all the characters typed in before the timeout are returned in the buffer. If omitted, characters are returned in the buffer until a terminator is seen.

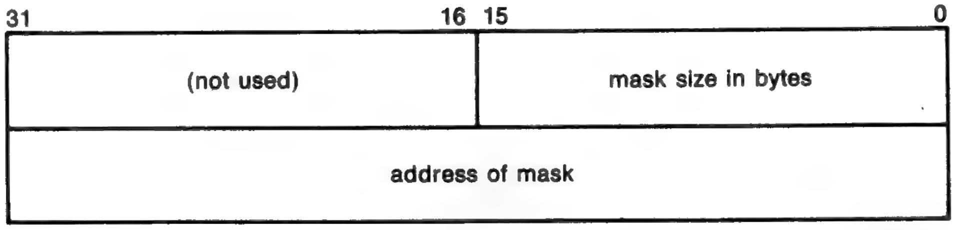
terminator-set

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor, fixed length**

Either a mask that specifies which characters are to be treated as terminators (short form) or a descriptor pointing to such a mask (long form). The **terminator-set** argument is the address of a descriptor pointing to the mask.

If you want to use terminators with ASCII values in the range 0 to 31, use the short form. You create this mask by setting the bit that corresponds to the ASCII value of desired terminator. For example, to specify that CTRL/A (ASCII value 1) is a terminator, you set bit 1 in the **terminator-set** mask.

If you want to use terminators with ASCII values outside the range 0 to 31, use the long form and create a descriptor of this form first:



ZK-2004-84

The mask itself has the same format as that of the short form; however, the long form allows use of a more comprehensive set of terminator characters. For example, a mask size of 16 bytes allows any 7-bit ASCII character to be set as a terminator, while a mask size of 32 bytes allows any 8-bit character to be set as a terminator. Any mask size between 1 and 32 bytes is acceptable.

If the terminator mask is all zeros, there are no specified terminators and the read terminates when the number of characters specified in the **max-length** argument have been transferred.

If the **terminator-set** argument is omitted, the set of terminators is the VMS default terminator set. For more information see the *VAX/VMS I/O User's Reference Manual: Part I*, Section 8.4.1.2.

ini-offset

VMS Usage: **longword_signed**
type: **longword (signed)**
access: **read only**
mechanism: **by reference**

Input string offset. The **ini-offset** argument is a longword that contains the number of characters (from the **in-string** argument) to output after the prompt before waiting for input.

terminator-code

VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

Key terminator code. The **terminator-code** argument is an unsigned word into which SMG$READ_VERIFY writes a code indicating what character or key terminated the read. Key terminator codes are of the form SMG$K_TRM_keyname. The keynames are listed in Table RTL-1, in Chapter 3.

display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Display identifier. The optional **display-id** argument is the address of an unsigned longword integer that contains the identifier of the virtual display in which the read is to be performed. If specified, SMG$READ_VERIFY begins the read at the current virtual cursor position in that virtual display. If omitted, the read begins in the current physical cursor position. Note that the length of the **prompt-string** plus the input is limited to the number of visible columns in the display.

alt-echo-string

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Alternate echo string. The **alt-echo-string** argument is a string that is printed after the first character is typed during the read operation. This is an optional argument.

alt-display-id

VMS Usage: **longword_signed**
type: **longword (signed)**
access: **read only**
mechanism: **by reference**

Alternate display identification. The **alt-display-id** is a signed longword integer containing the identification of the virtual display in which the **alt-echo-string** argument is to be printed. This is an optional argument. If specified, the output begins at the current virtual cursor position in that virtual display. If omitted, the value of the **display-id** argument is used as the default. If **display-id** is not specified, the output begins in the current physical cursor position.

rendition-set

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute specifier. The optional **rendition-set** argument is the address of a longword bit mask in which each 1-bit attribute causes the corresponding attribute to be set in the display. The following attributes can be specified using the **rendition-set** argument:

SMG$M_BLINK	Displays characters blinking.
SMG$M_BOLD	Displays characters in higher-than-normal intensity (bolded).
SMG$M_REVERSE	Displays characters in reverse video — that is, using the opposite default rendition of the virtual display.
SMG$M_UNDERLINE	Displays characters underlined.

The **display-id** argument must be specified when using the **rendition-set** argument.

rendition-complement

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Attribute complement specifier. The optional **rendition-complement** argument is the address of a longword bit mask in which each 1-bit attribute causes the corresponding attribute to be complemented in the display. All of the attributes that can be specified with the **rendition-set** argument can be complemented with the **rendition-complement** argument. The **display-id** argument must be specified when using the **rendition-complement** argument.

DESCRIPTION

This routine reads a sequence of characters from the virtual keyboard specified and verifies the sequence against the picture string. It then returns characters read to the caller. The caller may also specify that a code indicating the terminator be returned.

The optional arguments **rendition-set** and **rendition-complement** let the user control the attributes of the virtual display in which the read is done. The **rendition-set** argument sets certain virtual display attributes, while **rendition-complement** complements these attributes. If the same bit is specified in both the **rendition-set** and **rendition-complement** parameters, the **rendition-set** is evaluated first, followed by **rendition-complement**. By using these two parameters together, the user can control each virtual display attribute in a single procedure call. On a single-attribute basis, the user can cause the following transformations:

Set	Complement	Action
0	0	Attribute unchanged.
1	0	Attribute set to "on."
0	1	Attribute set to complement of current setting.
1	1	Attribute set to "off."

For additional information on read-verify operations, see the *VAX/VMS I/O User's Reference Manual: Part I*. Note that display batching for both the pasteboard and the virtual display must be off when you use SMG$READ_VERIFY.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SS$_CANCEL	I/O operation canceled while queued (by SMG$CANCEL_INPUT).
SS$_ABORT	I/O operation aborted during execution (by SMG$CANCEL_INPUT).
SMG$_DISREQ	A call to SMG$READ_VERIFY was made specifying right-justification; no **display-id** was specified; and the SCROLL_REVERSE sequence was not found for this terminal in TERMTABLE.EXE. Add the **display-id** argument to the SMG$READ_VERIFY call or add the SCROLL_REVERSE sequence to TERMTABLE.EXE.
SMG$_EOF	End-of-file.
SMG$_INVDIS_ID	Invalid **display-id**.
SMG$_INVKBD_ID	Invalid **keyboard-id**.
SMG$_LENNOTEQL	Length of **pic-string** and **in-string** are not equal.
SMG$_LENMUSONE	Length of **fill-char** and **clear-char** must be 1.
SMG$_WRONUMARG	Wrong number of arguments.
LIB$_xxx	Any error from LIB$SCOPY_R_DX.
RMS$_xxx	Any error from $GET (except RMS$_EOF).
SS$_xxx	Any error from $QIOW.

SMG$REPAINT_LINE Repaint One or More Lines on the Current Screen

SMG$REPAINT_LINE repaints a series of lines on the current screen.

FORMAT

SMG$REPAINT_LINE *pasteboard-id ,row-start [,num-of-lines]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

pasteboard-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Pasteboard identifier. The **pasteboard-id** argument is the address of the pasteboard associated with the physical screen to be repainted.

row-start

VMS Usage: **longword_signed**
type: **longword (signed)**
access: **read only**
mechanism: **by reference**

Starting row number. The **row-start** argument is the address of the physical row number to start repainting.

num-of-lines

VMS Usage: **longword_signed**
type: **longword (signed)**
access: **read only**
mechanism: **by reference**

Number of contiguous lines to repaint. The **num-of-lines** argument is the address of a signed longword containing the number of lines. This argument is optional. If not specified, the default is 1.

DESCRIPTION

SMG$REPAINT_LINE repaints a line or series of lines on the current screen based on its internal knowledge of what the screen should look like. You should call SMG$REPAINT_LINE when you suspect that the screen has been disrupted.

If pasteboard batching is in effect, the repaint occurs from the screen image; otherwise, it occurs from the text image. SMG$REPAINT_LINE has the added benefit of circumventing the restriction that the display you are working on must be pasted to column one. (For further information on this restriction, refer to the description section of SMG$READ_STRING.)

This routine should not be used if the line being repainted is double high.

One good use of SMG$REPAINT_LINE is to restore a line after entering a CTRL/U or CTRL/R to an input routine.

CONDITION VALUES SIGNALED

SS$NORMAL	Normal successful completion.
SMG$_INVPAS_ID	Invalid pasteboard control block.

SMG$REPAINT_SCREEN—Repaint Current Screen

SMG$REPAINT_SCREEN repaints the current screen after nonSMG I/O has occurred.

FORMAT

SMG$REPAINT_SCREEN *pasteboard-id*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENT

pasteboard-id

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Specifies the terminal screen to be repainted. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

DESCRIPTION

SMG$REPAINT_SCREEN repaints the current screen. It is intended to be used when some outside agent (for example, a broadcast message) has disrupted the screen.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVPAS_ID	Invalid **pasteboard-id**.
SMG$_WRONUMARG	Wrong number of arguments.

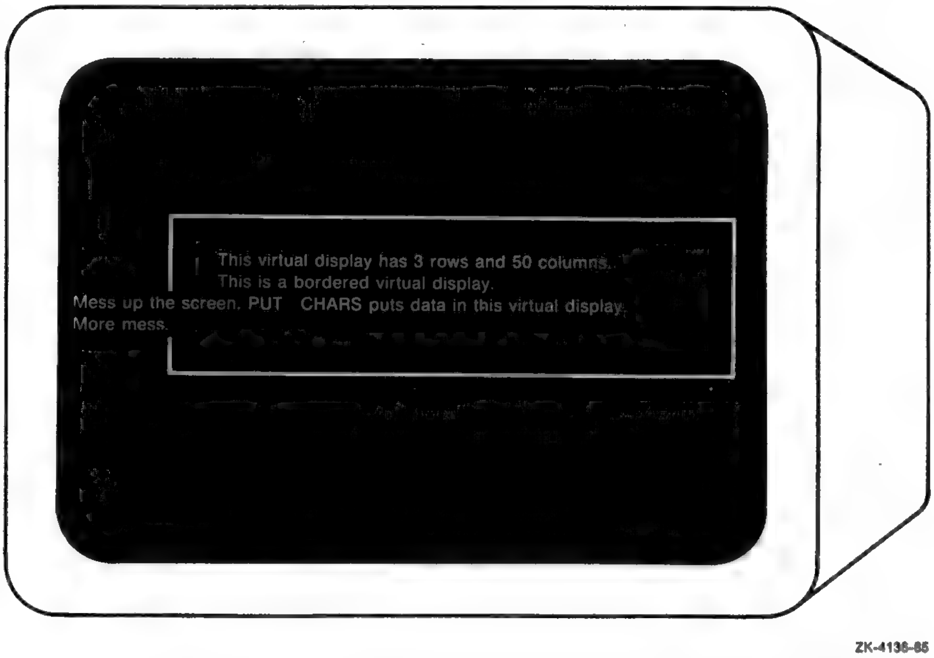
EXAMPLE

```
C+
C This FORTRAN example program demonstrates
C the use of SMG$REPAINT_SCREEN.
C-
        IMPLICIT INTEGER (A-Z)
C+
C Create the virtual display by calling
C SMG$CREATE_VIRTUAL_DISPLAY. To create
C a border, we set BORDER = 1. No border
C would be BORDER = 0.
C-
        INCLUDE '($SMGDEF)'
        ROWS = 3
        COLUMNS = 50
        BORDER = 1
        STATUS = SMG$CREATE_VIRTUAL_DISPLAY
```

```
     1          (ROWS, COLUMNS, DISPLAY1, BORDER)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Create the pasteboard using SMG$CREATE_PASTEBOARD.
C-
        STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Put data in the virtual display by calling SMG$PUT_CHARS.
C-
        STATUS = SMG$PUT_CHARS ( DISPLAY1,
     1      ' This virtual display has 3 rows and 50 columns.', 1, 1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        STATUS = SMG$PUT_CHARS ( DISPLAY1,
     1      ' This is a bordered virtual display.', 2, 1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        STATUS = SMG$PUT_CHARS ( DISPLAY1,
     1      ' SMG$PUT_CHARS puts data in this virtual display.', 3, 1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Call SMG$PASTE_VIRTUAL_DISPLAY to paste the virtual display.
C-
        STATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DISPLAY1, PASTE1, 4, 15)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Mess up the screen with some FORTRAN output.
C-
        WRITE (6,*) 'Mess up the screen.'
        WRITE (6,*) 'More mess.'
C+
C Call SMG$REPAINT_SCREEN to repaint the screen.
C-
        STATUS = SMG$REPAINT_SCREEN ( PASTE1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        END
```

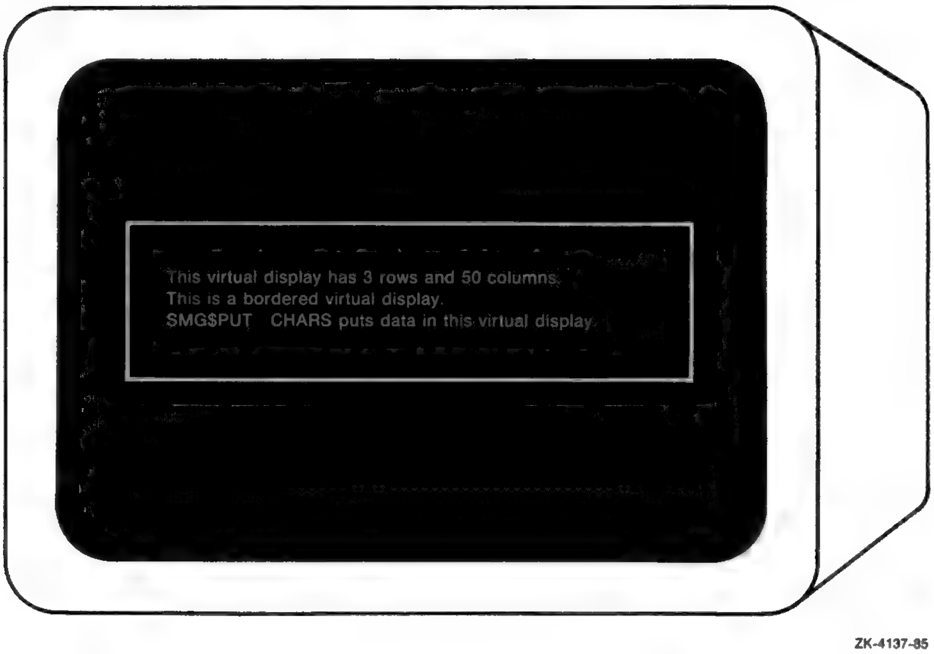
The output generated by this FORTRAN program before the call to SMG$REPAINT_SCREEN is shown in Figure RTL-36.

Figure RTL-36 Output Generated by FORTRAN Program Calling SMG$REPAINT_SCREEN



The output generated after the call to SMG$REPAINT_SCREEN is shown in Figure RTL-37.

Figure RTL-37 Output Generated by FORTRAN Program Calling SMG$REPAINT_SCREEN



SMG$REPASTE_VIRTUAL_DISPLAY Repaste Virtual Display

SMG$REPASTE_VIRTUAL_DISPLAY moves a virtual display to a new position on the pasteboard. The pasting order is not preserved.

FORMAT

SMG$REPASTE_VIRTUAL_DISPLAY *display-id ,pasteboard-id ,pb-row ,pb-column [,top-display-id]*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

display-id

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Specifies the virtual display to be repasted. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

Display-id is returned by SMG$CREATE_VIRTUAL_DISPLAY.

pasteboard-id

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Specifies the pasteboard on which the display is repasted. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

pb-row

VMS Usage:	**longword_signed**
type:	**longword integer (signed)**
access:	**read only**
mechanism:	**by reference**

Specifies the pasteboard row that is to contain row 1 of the specified virtual display. The **pb-row** argument is the address of a signed longword integer that contains the pasteboard row.

pb-column

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the pasteboard column that is to contain column 1 of the specified virtual display. The **pb-column** argument is the address of a signed longword integer that contains the pasteboard column.

top-display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Optional identifier of the virtual display under which **display-id** will be pasted. The **top-display-id** argument is the address of an unsigned longword containing the identifier of this virtual display. Note that the virtual display specified by **top-display-id** must already be pasted.

DESCRIPTION

SMG$REPASTE_VIRTUAL_DISPLAY lets you move a virtual display to a new position on its pasteboard. This routine calls SMG$UNPASTE_VIRTUAL_DISPLAY and SMG$PASTE_VIRTUAL_DISPLAY. Note that this changes the pasting order. The unpasting and repasting operations use the SMG$BEGIN_PASTEBOARD_UPDATE and SMG$END_PASTEBOARD_UPDATE; thus, there is no effect on the screen until the repasting operation is complete.

Note that this routine may cause the virtual display to be at the top of the pasting order. To move a virtual display without changing its pasting order, use SMG$MOVE_VIRTUAL_DISPLAY. If the optional argument **top-display-id** is specified, SMG$REPASTE_VIRTUAL_DISPLAY pastes the virtual display being repasted under the virtual display specified by **top-display-id**. In this case, the virtual display specified by **top-display-id** must already be pasted.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVDIS_ID	Invalid **display-id**.
SMG$_INVPAS_ID	Invalid **pasteboard-id**.
SMG$_WRONUMARG	Wrong number of arguments.

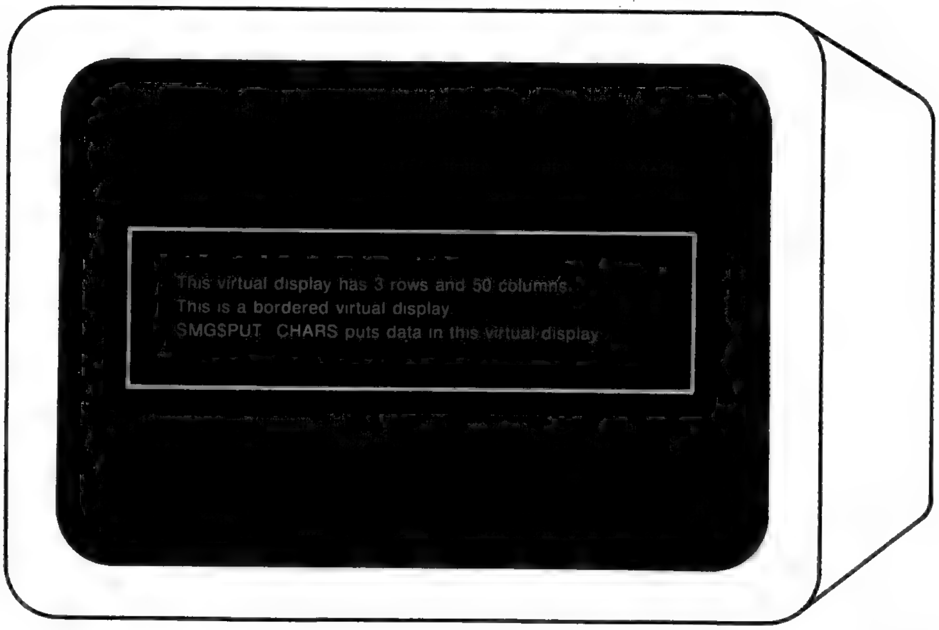
EXAMPLE

```
C+
C This FORTRAN example program demonstrates the use of
C SMG$REPASTE_VIRTUAL_DISPLAY and SMG$MOVE_VIRTUAL_DISPLAY.
C-
        IMPLICIT INTEGER (A-Z)
C+
C Include the SMG definitions. In particular, we want SMG$M_BORDER.
C-
        INCLUDE '($SMGDEF)'
C+
C Create a virtual display with a border by calling
```

```
C SMG$CREATE_VIRTUAL_DISPLAY.
C-
        ROWS = 3
        COLUMNS = 50
        STATUS = SMG$CREATE_VIRTUAL_DISPLAY
     1          (ROWS, COLUMNS, DISPLAY1, SMG$M_BORDER)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Call SMG$CREATE_PASTEBOARD to create the pasteboard.
C-
        STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Put data in the virtual display using SMG$PUT_CHARS.
C-
        STATUS = SMG$PUT_CHARS ( DISPLAY1,
     1      ' This virtual display has 3 rows and 50 columns.', 1, 1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        STATUS = SMG$PUT_CHARS ( DISPLAY1,
     1      ' This is a bordered virtual display.', 2, 1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        STATUS = SMG$PUT_CHARS ( DISPLAY1,
     1      ' SMG$PUT_CHARS puts data in this virtual display.', 3, 1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Call SMG$PASTE_VIRTUAL_DISPLAY to paste the virtual display.
C-
        STATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DISPLAY1, PASTE1, 4, 15)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Move the virtual display by calling SMG$MOVE_VIRTUAL_DISPLAY.
C-
        STATUS = SMG$MOVE_VIRTUAL_DISPLAY ( DISPLAY1, PASTE1, 10, 5)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
C+
C Call SMG$REPASTE_VIRTUAL_DISPLAY to repaste the
C original virtual display as it was.
C-
        STATUS = SMG$REPASTE_VIRTUAL_DISPLAY (DISPLAY1, PASTE1, 4, 15)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        END
```

The output generated by this FORTRAN program before the call to SMG$MOVE_VIRTUAL_DISPLAY is shown in Figure RTL-38.

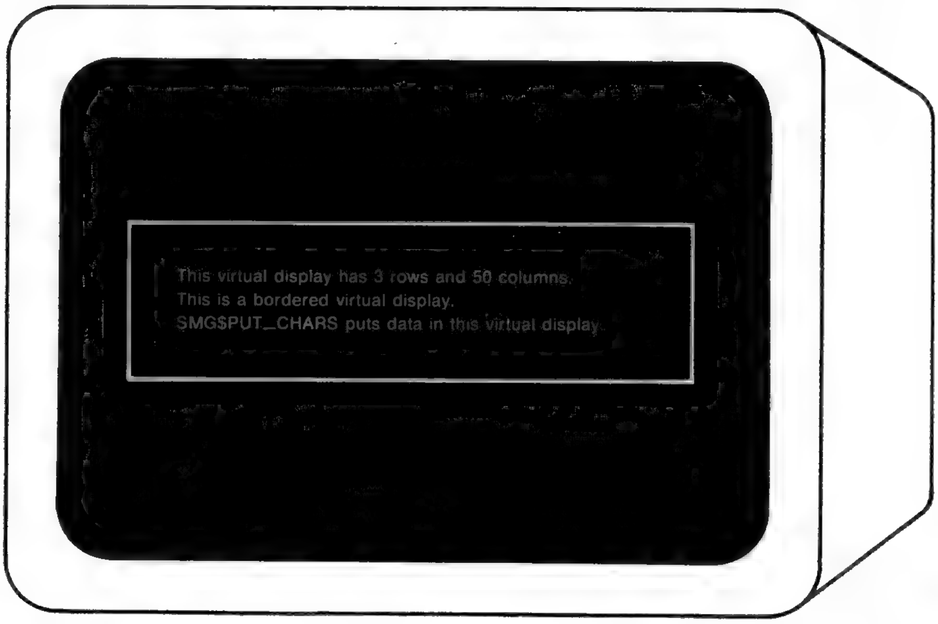
Figure RTL-38 Output before the Call to SMG$MOVE_VIRTUAL_DISPLAY



ZK-4139-85

After the call to SMG$MOVE_VIRTUAL_DISPLAY, the output shown is that illustrated in Figure RTL-39.

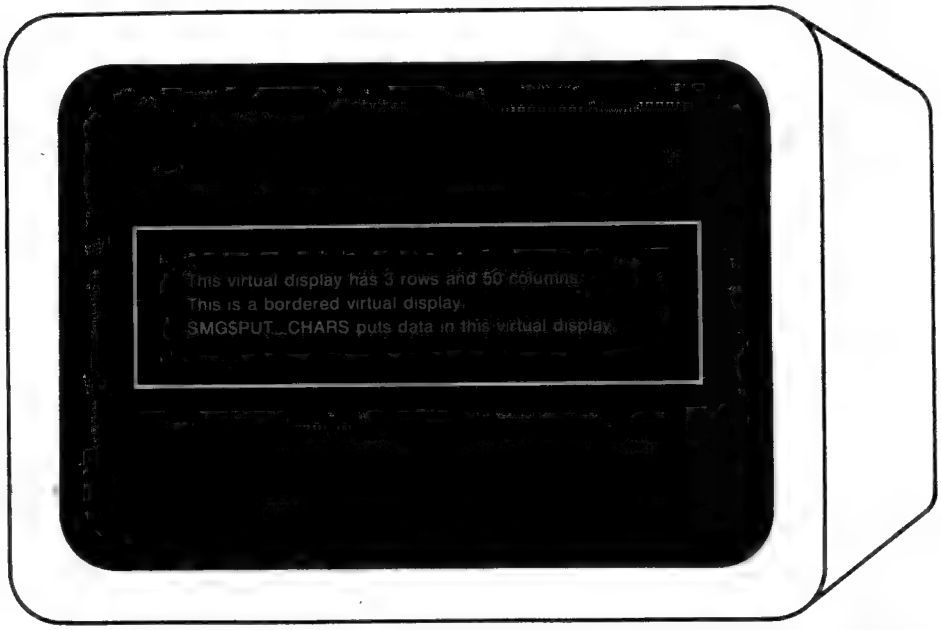
Figure RTL-39 Output Displayed After the Call to SMG$MOVE_VIRTUAL_DISPLAY



ZK-4141-85

Figure RTL-40 shows the final output displayed, after the call to SMG$REPASTE_VIRTUAL_DISPLAY.

Figure RTL-40 Output Displayed After the Call to SMG$REPASTE_VIRTUAL_DISPLAY



ZK-4130-85

SMG$REPLACE_INPUT_LINE—Replace Input Line

SMG$REPLACE_INPUT_LINE replaces the specified lines in the recall buffer with the specified string.

FORMAT

SMG$REPLACE_INPUT_LINE *keyboard-id [,out-line] [,num-of-lines]*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

keyboard-id

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Keyboard identifier. The **keyboard-id** argument is the address of an unsigned longword containing the identifier of the virtual keyboard from which to read. The virtual keyboard is created by calling the SMG$CREATE_VIRTUAL_KEYBOARD routine.

out-line

VMS Usage:	**char_string**
type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

String that contains the line to be entered into the recall buffer. The **out-line** argument is the address of a descriptor pointing to this string. The default is a null string, which removes the last line entered.

num-of-lines

VMS Usage:	**byte_unsigned**
type:	**byte (unsigned)**
access:	**read only**
mechanism:	**by reference**

Number of lines to be replaced. The **num-of-lines** argument is the address of an unsigned byte containing the number of lines to be replaced with **out-line**. The default value for the **num-of-lines** argument is 1 (the last line entered).

DESCRIPTION

SMG$REPLACE_INPUT_LINE replaces the requested lines in the recall buffer with the specified string. The remaining (**num-of-lines** - 1) lines are deleted. This routine is intended to aid in processing line continuations.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVKBD_ID	Invalid **keyboard-id**.
SMG$_WRONUMARG	Wrong number of arguments.
LIB$_INSVIRMEM	Insufficient virtual memory.

SMG$RESTORE_PHYSICAL_SCREEN Restore Physical Screen

SMG$RESTORE_PHYSICAL_SCREEN rewrites the screen image as it was at the time the SMG$SAVE_PHYSICAL_SCREEN routine was called.

FORMAT

SMG$RESTORE_PHYSICAL_SCREEN *pasteboard-id ,saved-display-id*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

pasteboard-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the physical screen to be restored. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

Pasteboard-id is returned by the SMG$CREATE_PASTEBOARD routine.

saved-display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the virtual display created by the SMG$SAVE_PHYSICAL_SCREEN routine. The **saved-display-id** argument is the address of an unsigned longword that contains this display identifier.

DESCRIPTION

SMG$RESTORE_PHYSICAL_SCREEN reproduces the screen image saved by the SMG$SAVE_PHYSICAL_SCREEN routine. You must pass the **display-id** returned by the SMG$SAVE_PHYSICAL_SCREEN routine to the SMG$RESTORE_PHYSICAL_SCREEN routine. Note that when performing multiple calls to SMG$SAVE_PHYSICAL_SCREEN and SMG$RESTORE_PHYSICAL_SCREEN, the calls must be performed in a nested fashion; that is, the last screen saved must be the first one restored.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVDIS_ID	Invalid **display-id**.
SMG$_INVPAS_ID	Invalid **pasteboard-id**.

SMG$RETURN_CURSOR_POS—Return Cursor Position

SMG$RETURN_CURSOR_POS returns the current virtual cursor position in a specified virtual display.

FORMAT

SMG$RETURN_CURSOR_POS *display-id ,row-number ,column-number*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the virtual display whose current cursor position you are requesting. The **display-id** argument is the address of an unsigned longword that contains the display identifier. **Display-id** is returned by SMG$CREATE_VIRTUAL_DISPLAY.

row-number

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **write only**
mechanism: **by reference**

Receives the cursor's current row position within the specified virtual display. The **row-number** argument is the address of a longword into which is written the current row position.

column-number

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **write only**
mechanism: **by reference**

Receives the cursor's current column position within the specified virtual display. The **column-number** argument is the address of a longword into which is written the current column position.

DESCRIPTION

SMG$RETURN_CURSOR_POS returns the virtual cursor's current row and column positions in a specified virtual display.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVDIS_ID	Invalid **display-id**.
SMG$_WRONUMARG	Wrong number of arguments.

SMG$RETURN_INPUT_LINE—Return Input Line

SMG$RETURN_INPUT_LINE returns to the caller the requested line from the recall buffer. This line is retrieved either by matching it with a specified string or by specifying the appropriate line number.

FORMAT

SMG$RETURN_INPUT_LINE *keyboard-id ,out-line [,match-string] [,line-num] [,out-length]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

keyboard-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Keyboard identifier. The **keyboard-id** argument is the address of an unsigned longword containing the identifier of the virtual keyboard from which to read. The virtual keyboard is created by calling the SMG$CREATE_VIRTUAL_KEYBOARD routine.

out-line

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

String into which is written the complete recalled line. The **out-line** argument is the address of a descriptor pointing to this string.

match-string

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Match string to be used when searching for the line to be recalled. The optional **match-string** argument is the address of a descriptor pointing to this match string. The search begins with the last line typed.

line-num

VMS Usage: **byte_unsigned**
type: **byte (unsigned)**
access: **read only**
mechanism: **by reference**

Line number to be used when searching for the line to be recalled. The optional **line-num** argument is the address of an unsigned byte containing the number of the line to be recalled. The last line typed is line number 1.

out-length

VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

Length of the **out-line** string. The optional **out-length** argument is the address of an unsigned word containing either the number of characters read or the maximum length of **out-line**, whichever is less.

DESCRIPTION

SMG$RETURN_INPUT_LINE returns to the caller the specified line in the recall buffer. If the **match-string** argument is specified, SMG$RETURN_INPUT_LINE searches for and returns the line that matches the specified string. If the **line-num** argument is specified, SMG$RETURN_INPUT_LINE returns the line that corresponds to the specified line number. This routine is intended to aid in the implementation of a DCL style "RECALL" command.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVKBD_ID	Invalid **keyboard-id**.
SMG$_WRONUMARG	Wrong number of arguments.
LIB$_xxx	Any error from LIB$COPY_R_DX.

SMG$RING_BELL—Ring the Terminal Bell or Buzzer

SMG$RING_BELL sounds the terminal bell or buzzer.

FORMAT

SMG$RING_BELL *display-id [,number-of-times]*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

display-id

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Specifies the virtual display for which the bell or buzzer sounds. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

Display-id is returned by SMG$CREATE_VIRTUAL_DISPLAY.

number-of-times

VMS Usage:	**longword_signed**
type:	**longword integer (signed)**
access:	**read only**
mechanism:	**by reference**

Specifies the number of times the bell or buzzer is sounded. The **number-of-times** argument is the address of a signed longword integer that contains the number of times the bell or buzzer is sounded. If omitted, 1 is used.

DESCRIPTION

SMG$RING_BELL sounds the bell or buzzer on each pasteboard (terminal) to which the specified virtual display is pasted. The bell or buzzer sounds the number of times specified; the default number of times is 1.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVDIS_ID	Invalid **display-id**.

Any condition values returned by $QIOW.

Any condition values returned by STR$DUPL_CHAR.

SMG$SAVE_PHYSICAL_SCREEN
Save Physical Screen

SMG$SAVE_PHYSICAL_SCREEN saves the contents of the screen so that a later call to SMG$RESTORE_PHYSICAL_SCREEN can restore it.

FORMAT

SMG$SAVE_PHYSICAL_SCREEN
pasteboard-id ,saved-display-id
[,desired-row-start]
[,desired-row-end]

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

pasteboard-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the physical screen whose contents are to be saved. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

saved-display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Receives the display id of the display created to contain the contents of the physical screen. The **saved-display-id** argument is the address of an unsigned longword into which the display identifier is written.

Saved-display-id must be passed to the SMG$RESTORE_PHYSICAL_SCREEN routine to restore the saved information.

desired-row-start

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the first row to be saved. The **desired-row-start** argument is the address of a signed longword integer that contains the row number. If omitted, row 1 of the pasteboard is used.

desired-row-end

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the last row to be saved. The **desired-row-end** argument is the address of a signed longword integer that contains the row number. If omitted, the last row of the pasteboard is used.

DESCRIPTION

SMG$SAVE_PHYSICAL_SCREEN blanks the screen by creating a virtual display that is as wide as the physical screen and as high as specified by the **desired-row-start** and **desired-row-end** arguments. If these two arguments are omitted, the created virtual display is as high as the physical screen. The information saved — that is, the screen image — can be restored by calling the SMG$RESTORE_PHYSICAL_SCREEN routine. When performing multiple calls to SMG$SAVE_PHYSICAL_SCREEN and SMG$RESTORE_PHYSICAL_SCREEN, the calls must be performed in a nested order; that is, the last screen saved must be the first one restored, and so on.

These routines are useful when calling a procedure that may send output to the screen without using the Screen Management Facility. Before calling such a procedure, you save the screen image with SMG$SAVE_PHYSICAL_SCREEN. After the procedure executes, you restore the screen image with SMG$RESTORE_PHYSICAL_SCREEN.

Note that when using SMG$SAVE_PHYSICAL_SCREEN on a terminal that does not support scrolling regions, you must save and restore the entire screen.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVPAS_ID	Invalid **pasteboard-id**.
SMG$_WRONUMARG	Wrong number of arguments.
LIB$_INSVIRMEM	Insufficient virtual memory.

SMG$SCROLL_DISPLAY_AREA—Scroll Display Area

SMG$SCROLL_DISPLAY_AREA scrolls a rectangular region of a virtual display.

FORMAT

SMG$SCROLL_DISPLAY_AREA *display-id [,starting-row ,starting-column] [,height] [,width] [,direction] [,count]*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

display-id

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Specifies the virtual display in which scrolling takes place. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

Display-id is returned by SMG$CREATE_VIRTUAL_DISPLAY.

starting-row

VMS Usage:	**longword_signed**
type:	**longword integer (signed)**
access:	**read only**
mechanism:	**by reference**

Specifies the first row of the scrolling region. The **starting-row** argument is the address of a signed longword integer that contains the starting row.

If omitted, row 1 of the specified virtual display is used. Note that if you omit either **starting-row** or **starting-column**, the default (row 1 and column 1) is used.

starting-column

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the first column of the scrolling region. The **starting-column** argument is the address of a signed longword integer that contains the starting column.

If omitted, column 1 of the specified virtual display is used. Note that if you omit either **starting-row** or **starting-column**, the default (row 1 and column 1) is used.

height

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the number of rows in the scrolling region. The **height** argument is the address of a signed longword integer that contains the number of rows.

If omitted, this value defaults to either the height of the scrolling region (if one has been explicitly set with SMG$SET_DISPLAY_SCROLL_REGION) or the height of the specified virtual display.

width

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the number of columns in the scrolling region. The **width** argument is the address of a signed longword integer that contains the number of columns.

If omitted, this value defaults to either the width of the scrolling region (if one has been explicitly set with SMG$SET_DISPLAY_SCROLL_REGION) or the width of the specified virtual display.

direction

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the direction to scroll. The **direction** argument is the address of an unsigned longword that contains the direction code.

Valid values are SMGM_UP, SMGM_DOWN, SMG$M_RIGHT and SMG$M_LEFT. SMG$M_UP is the default.

count

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the number of lines to scroll. The **count** argument is the address of a signed longword integer that contains the number of lines to scroll. If omitted, one line is scrolled.

DESCRIPTION

SMG$SCROLL_DISPLAY_AREA scrolls a rectangular region of the specified virtual display. It scrolls the region a specified number of lines in the specified direction.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVDIS_ID	Invalid **display-id**.
SMG$_INVCOL	Invalid column.
SMG$_INVROW	Invalid row.
SMG$_WRONUMARG	Wrong number of arguments.

SMG$SET_BROADCAST_TRAPPING Enable Broadcast Trapping

SMG$SET_BROADCAST_TRAPPING enables the trapping of broadcast messages.

FORMAT

SMG$SET_BROADCAST_TRAPPING *pasteboard-id [,AST-routine] [,AST-argument]*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

pasteboard-id

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Specifies the pasteboard for the terminal to be affected. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

Pasteboard-id is returned by SMG$CREATE_PASTEBOARD.

AST-routine

VMS Usage:	**ast_procedure**
type:	**procedure entry mask**
access:	**read only**
mechanism:	**by reference**

The address of an AST routine to be called when a message is received at the terminal. The **AST-routine** argument is the address of the routine's procedure entry mask — that is, the address of the routine itself.

When the **AST-routine** argument is either omitted or is given a value of zero, the BROADCAST mode is set to synchronize. In this mode, you must periodically call SMG$GET_BROADCAST_MESSAGE to see if any broadcast messages have arrived.

AST-argument

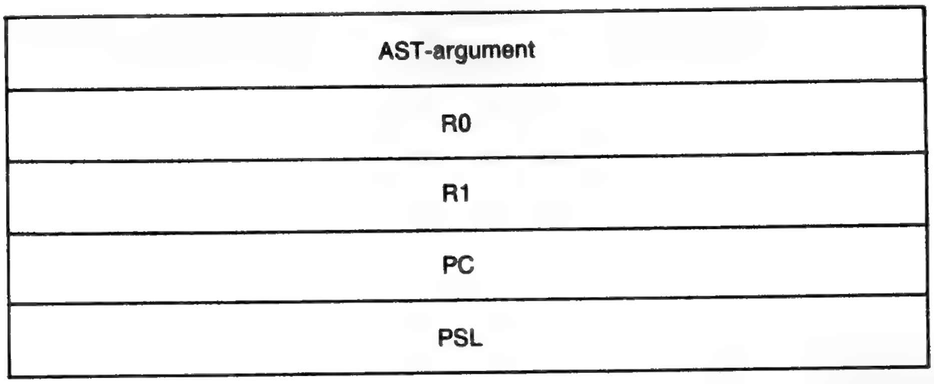
VMS Usage:	**user_arg**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by value**

A value to be passed to the AST routine. **AST-argument** is the value to be passed to the AST routine.

DESCRIPTION

SMG$SET_BROADCAST_TRAPPING enables the trapping of broadcast messages sent to the specified pasteboard (terminal). If you enable broadcast trapping with SMG$SET_BROADCAST_TRAPPING but do not disable it with SMG$DISABLE_BROADCAST_TRAPPING before the image exits, any messages that have been broadcast to the terminal are lost when the image exits.

The AST routine is called with 5 parameters: **AST-argument**, R0, R1, PC, and PSL.



ZK-4803-85

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVPAS_ID	Invalid **pasteboard-id**.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_NOT_A_TRM	Informational message. The pasteboard is not a terminal.

Any condition values returned by $DASSGN or $CANCEL.

Any condition values returned by LIB$ASN_WTH_MBX.

EXAMPLE

For an example of using SMG$SET_BROADCAST_TRAPPING, see the example for the routine SMG$DISABLE_BROADCAST_TRAPPING.

SMG$SET_CURSOR_ABS—Set Absolute Cursor Position

SMG$SET_CURSOR_ABS moves the virtual cursor to the specified position in a virtual display.

FORMAT

SMG$SET_CURSOR_ABS *display-id [,row] [,column]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the virtual display in which to set the virtual cursor position. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

Display-id is returned by SMG$CREATE_VIRTUAL_DISPLAY.

row

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the row position to which the virtual cursor moves. The **row** argument is the address of a signed longword integer that contains the row number. If omitted, the cursor remains at the current row.

column

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the column position to which the virtual cursor moves. The **column** argument is the address of a signed longword integer that contains the column number. If omitted, the virtual cursor remains at the current column.

DESCRIPTION

SMG$SET_CURSOR_ABS moves the virtual cursor to the specified position in the specified virtual display.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVDIS_ID	Invalid **display-id**.
SMG$_INVCOL	Invalid column.
SMG$_INVROW	Invalid row.
SMG$_WRONUMARG	Wrong number of arguments.

SMG$SET_CURSOR_MODE
Turn the Physical Cursor On or Off

SMG$SET_CURSOR_MODE turns the physical cursor on or off.

FORMAT

SMG$SET_CURSOR_MODE *pasteboard-id ,cursor-mode*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

pasteboard-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Pasteboard identifier. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

cursor-mode

VMS Usage: **boolean**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Longword that determines whether or not the physical cursor is displayed. The **cursor-mode** argument is the address of an unsigned longword that determines the status of the physical cursor: if **cursor-mode** is zero, the physical cursor is displayed; if **cursor-mode** is set to 1, the physical cursor is invisible.

DESCRIPTION

SMG$SET_CURSOR_MODE turns the cursor on and off. The cursor is displayed if the **cursor-mode** argument is zero. If the **cursor-mode** argument is set to 1, the cursor is invisible. If your terminal does not have this capability defined, this routine has no effect.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_INVPAS_ID	Invalid **pasteboard-id**.
SMG$_INVARG	Invalid argument.

SMG$SET_CURSOR_REL—Move Cursor Relative To Current Position

SMG$SET_CURSOR_REL moves the virtual cursor the specified number of rows and columns from the current virtual cursor position in a virtual display.

FORMAT

SMG$SET_CURSOR_REL *display-id [,delta-row] [,delta-column]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the virtual display in which to move the virtual cursor. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

Display-id is returned by SMG$CREATE_VIRTUAL_DISPLAY.

delta-row

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the number of rows to move the virtual cursor. The **delta-row** argument is the address of a signed longword integer that contains the number of rows to move. If omitted, the virtual cursor remains at the current row position. If **delta-row** is positive, the virtual cursor moves downward the specified number of rows. If **delta-row** is negative, the virtual cursor moves upward the specified number of rows.

delta-column

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the number of columns to move the cursor. The **delta-column** argument is the address of a longword that contains the number of columns to move. If omitted, the virtual cursor remains at the current column position. If **delta-column** is positive, the virtual cursor moves the specified number of columns to the right. If **delta-column** is negative, the virtual cursor moves the specified number of columns to the left.

DESCRIPTION

SMG$SET_CURSOR_REL moves the virtual cursor the specified number of rows and columns relative to the current virtual cursor position. If the specified **delta-row** or **delta-column** causes the cursor to move outside the bounds of the virtual display, SMG$_INVROW or SMG$_INVCOL is returned.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVDIS_ID	Invalid **display-id**.
SMG$_INVARG	Invalid argument.
SMG$_INVCOL	An invalid value of **delta-column** caused the cursor to move outside the bounds of the virtual display.
SMG$_INVROW	An invalid value of **delta-row** caused the cursor to move outside the bounds of the virtual display.
SMG$_WRONUMARG	Wrong number of arguments.

SMG$SET_DEFAULT_STATE—Set Default State

SMG$SET_DEFAULT_STATE sets and/or returns the current default state for a key table.

FORMAT

SMG$SET_DEFAULT_STATE *key-table-id [,new-state] [,old-state]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

key-table-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read-only**
mechanism: **by reference**

Specifies the key table in which you are setting or inquiring about a default state. The **key-table-id** argument is the address of an unsigned longword that contains the key table identifier.

Key-table-id is returned by the SMG$CREATE_KEY_TABLE routine.

new-state

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Specifies the new default state for the entire key table. The **new-state** argument is the address of a descriptor pointing to the new state string. The specified state name is converted to uppercase and stripped of trailing blanks before use.

old-state

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

Receives the existing default state name of the specified key definition table. The **old-state** argument is the address of a descriptor pointing to the storage into which the old state string is written.

DESCRIPTION

SMG$SET_DEFAULT_STATE sets and/or returns the default state name for an entire key definition table. By changing the default state for an entire key definition table, you can use the keypad keys for a new set of functions.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVKTB_ID	Invalid **key-table-id**.
SMG$_INVSTANAM	Invalid state name.
LIB$_INVSTRDES	Invalid string descriptor.

SMG$SET_DISPLAY_SCROLL_REGION Create Display Scrolling Region

SMG$SET_DISPLAY_SCROLL_REGION creates a scrolling region in a virtual display.

FORMAT

SMG$SET_DISPLAY_SCROLL_REGION *display-id [,starting-line] [,ending-line]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the virtual display in which scrolling takes place. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

Display-id is returned by SMG$CREATE_VIRTUAL_DISPLAY.

starting-line

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the first line of the scrolling region. The **starting-line** argument is the address of a signed longword integer that contains the starting line number. If omitted, the first line of the display is used.

ending-line

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Specifies the last line of the scrolling region. The **ending-line** argument is the address of a signed longword integer that contains the ending line number. If omitted, the last line of the virtual display is used.

DESCRIPTION

SMG$SET_DISPLAY_SCROLL_REGION creates a logical scrolling region in a specified virtual display, using the specified starting and ending lines. If the **starting-line** and **ending-line** arguments are omitted, the entire display becomes a scrolling region. This routine does not change the appearance of the screen or the cursor position.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVDIS_ID	Invalid **display-id**.
SMG$_INVARG	**Ending-line** is less than or equal to **starting-line**.
SMG$_INVROW	Invalid row.
SMG$_WRONUMARG	Wrong number of arguments.

SMG$SET_KEYPAD_MODE—Set Keypad Mode

SMG$SET_KEYPAD_MODE sets the terminal's numeric keypad to either numeric or applications mode.

FORMAT

SMG$SET_KEYPAD_MODE *keyboard-id ,new-mode*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

keyboard-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the virtual keyboard whose mode is to be changed. The **keyboard-id** argument is the address of an unsigned longword that contains the keyboard identifier.

Keyboard-id is returned by SMG$CREATE_VIRTUAL_KEYBOARD.

new-mode

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies whether the keypad is to be in applications or numeric mode. The **new-mode** argument is the address of an unsigned longword that contains the new mode setting. If the low-order bit is clear, the keypad is set to numeric mode; if the low-order bit is set, the keypad is set to applications mode. All other bits must be zero.

DESCRIPTION

SMG$SET_KEYPAD_MODE sets the terminal's numeric keypad to either numeric or applications mode. In applications mode, numeric keypad keys are considered function keys and may be used as terminators. In numeric mode, these keys are equivalent to the corresponding keys on the main keyboard.

To enable a successful call to SMG$SET_KEYPAD_MODE, the terminal must support applications mode.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_INVKBD_ID	Invalid **keyboard-id**.

SMG$SET_OUT_OF_BAND_ASTS
Set Out-of-Band ASTs

SMG$SET_OUT_OF_BAND_ASTS either enables or disables the trapping of out-of-band characters.

FORMAT

SMG$SET_OUT_OF_BAND_ASTS *pasteboard-id ,control-char-mask ,AST-routine [,AST-argument]*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

pasteboard-id

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Specifies the terminal for which out-of-band characters are enabled or disabled. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier. **Pasteboard-id** is returned by SMG$CREATE_PASTEBOARD.

control-char-mask

VMS Usage:	**mask_longword**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Specifies which control characters are to be the new out-of-band control characters. The **control-char-mask** argument is the address of an unsigned longword that contains the mask. You create this mask by setting the bit that corresponds to the ASCII value of the desired character. For example, to specify that CTRL/A (ASCII value 1) is an out-of-band control character, you set bit 1 in the **control-char-mask**. If no bits are set in this mask, then no out-of-band ASTs occur. For more information see the *VAX/VMS I/O User's Reference Manual: Part I*, Section 8.4.3.5.

AST-routine

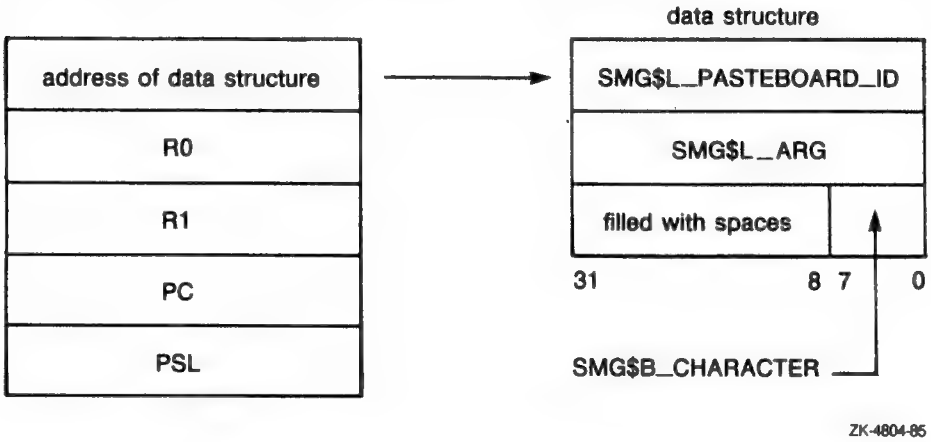
VMS Usage:	**ast_procedure**
type:	**procedure entry mask**
access:	**read only**
mechanism:	**by reference**

The address of an AST routine to be called when an out-of-band control character is typed at the terminal. The **AST-routine** argument is the address of the routine's procedure entry mask — that is, the address of the routine itself.

AST-argument

VMS Usage: **user_arg**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

The argument you supply for the AST. The **AST-argument** argument is an unsigned longword that contains the value to be passed to the AST routine. However, the AST routine may also need to know the out-of-band character and the **pasteboard-id** at which it was typed. Therefore, the Screen Management Facility creates a 3-longword structure to hold this information and passes the address of this structure as the first argument to the AST routine. The remaining four arguments are R0, R1, PC, and PSL. The Screen Management Facility stores the argument you supply in this structure.



The first longword contains the **pasteboard-id** and has the symbolic name SMG$L_PASTEBOARD_ID. The second longword contains the **AST-argument** and has the symbolic name SMG$L_ARG. The third longword contains the ASCII value of the out-of-band character typed and can be accessed by way of two symbolic names: SMG$B_CHARACTER (the low-order byte containing the ASCII value) and SMG$L_CHARACTER (the longword containing the ASCII value in the low-order byte and spaces in the high-order bytes).

DESCRIPTION

SMG$SET_OUT_OF_BAND_ASTS enables or disables the acceptance of out-of-band characters at the specified terminal. If one of these characters is typed at the terminal, the AST routine is called.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_INVPAS_ID	Invalid **pasteboard-id**.

SMG$SET_PHYSICAL_CURSOR Set Cursor on Physical Screen

SMG$SET_PHYSICAL_CURSOR moves the physical cursor to the specified position on the physical screen.

FORMAT

SMG$SET_PHYSICAL_CURSOR *pasteboard-id ,pb-row ,pb-column*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

pasteboard-id

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Specifies the physical screen whose physical cursor is to move. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

Pasteboard-id is returned by SMG$CREATE_PASTEBOARD.

pb-row

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Specifies the row to which the physical cursor moves. The **pb-row** argument is the address of an unsigned longword that contains the row number.

pb-column

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Specifies the column to which the physical cursor moves. The **pb-column** argument is the address of an unsigned longword that contains the column number.

DESCRIPTION

SMG$SET_PHYSICAL_CURSOR moves the physical cursor to the specified row and column position on a terminal screen.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_INVPAS_ID	Invalid **pasteboard-id**.
SMG$_INVARG	Invalid column.

SMG$SNAPSHOT—Write Snapshot

SMG$SNAPSHOT writes the current pasteboard buffer to the file or hardcopy terminal specified by **pasteboard-id**.

FORMAT

SMG$SNAPSHOT *pasteboard-id [,ff-flag]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENT

pasteboard-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the file or hardcopy terminal to receive the contents of the pasteboard buffer. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier. The output device assiocated with **pasteboard-id** is specified by the **output-device** argument of SMG$CREATE_PASTEBOARD.

ff-flag

VMS Usage: **boolean**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Form-feed flag. The **ff-flag** argument is the address of an unsigned longword containing a Boolean value. If the value of **ff-flag** is 1, then the first record output will be a form feed (<FF>). If the value of **ff-flag** is zero, then an initial form-feed record will not be output.

DESCRIPTION

SMG$SNAPSHOT is meant to be used primarily when output to the terminal is controlled by RMS — that is, when the output device is a file, a hardcopy terminal, or a terminal of unknown type. In this case, the pasteboard information is stored internally and is sent to either the file, hardcopy terminal, or the terminal of unknown type whenever SMG$SNAPSHOT is called. This allows you to capture screenlike images in a file.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_NOTRMSOUT	Successful completion. No action was taken because output is not controlled by RMS.

Any condition value returned by RMS.

SMG$UNPASTE_VIRTUAL_DISPLAY Remove Virtual Display

SMG$UNPASTE_VIRTUAL_DISPLAY removes a virtual display from a pasteboard.

FORMAT

SMG$UNPASTE_VIRTUAL_DISPLAY *display-id ,pasteboard-id*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

display-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the virtual display to be removed from a pasteboard. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

Display-id is returned by SMG$CREATE_VIRTUAL_DISPLAY.

pasteboard-id

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Specifies the pasteboard (physical screen) from which the virtual display is removed. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

DESCRIPTION

SMG$UNPASTE_VIRTUAL_DISPLAY removes the specified display from the specified pasteboard, and thus from the screen associated with the pasteboard. This routine does not destroy the virtual display or its contents; it merely removes its association with a particular pasteboard and hence its visibility on the screen. Any text that was occluded by the specified virtual display becomes visible again.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SMG$_INVPAS_ID	Invalid **pasteboard-id**.
SMG$_INVDIS_ID	Invalid **display-id**.
SMG$_WRONUMARG	Wrong number of arguments.
SMG$_INVARG	Invalid argument. The specified virtual display is not pasted to the specified pasteboard.
SMG$_NOTPASTED	The specified virtual display is not pasted to the specified pasteboard.

STR$ADD—Add Two Decimal Strings

STR$ADD adds two strings of digits.

FORMAT

STR$ADD *asign ,aexp ,adigits ,bsign ,bexp ,bdigits ,csign ,cexp ,cdigits*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

asign

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Sign of the first operand. The **asign** argument is the address of an unsigned longword containing this sign. Zero is considered positive; 1 is considered negative.

aexp

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Power of 10 by which **adigits** has to be multiplied to get the absolute value of the first operand. The **aexp** argument is the address of a signed longword integer containing this exponent.

adigits

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

String of unsigned digits representing the absolute value of the first operand before **aexp** is applied. The **adigits** argument is the address of a descriptor pointing to this string. This string must be an unsigned decimal number.

bsign

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Sign of the second operand. The **bsign** argument is the address of an unsigned longword containing the second operand's sign. Zero is considered positive; one is considered negative.

bexp

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Power of 10 by which **bdigits** has to be multiplied to get the absolute value of the second operand. The **bexp** argument is the address of a signed longword integer containing the second operand's exponent.

bdigits

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

String of unsigned digits representing the absolute value of the second operand before **bexp** is applied. The **bdigits** argument is the address of a descriptor pointing to this string. This string must be an unsigned decimal number.

csign

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **write only**
mechanism: **by reference**

Sign of the result. The **csign** argument is the address of a signed longword integer containing the result's sign. Zero is considered positive.

cexp

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **write only**
mechanism: **by reference**

Power of 10 by which **cdigits** has to be multiplied to get the absolute value of the result. The **cexp** argument is the address of a signed longword integer containing the result's exponent.

cdigits

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

String of unsigned digits representing the absolute value of the result before **cexp** is applied. The **cdigits** argument is the address of a descriptor pointing to this string. This string is an unsigned decimal number.

DESCRIPTION

STR$ADD adds two strings of decimal numbers (**a** and **b**). Each number to be added is passed to STR$ADD in three arguments:

1. **xdigits**—the string portion of **x**
2. **xexp**—the power of ten needed to obtain the absolute value of **x**
3. **xsign**—the sign of **x**

The value of the number x is derived by multiplying **xdigits** by 10**xexp** and applying **xsign**. Therefore, if **xdigits** is equal to '2' and **xexp** is equal to 3 and **xsign** is equal to 1, then the number represented in the **x** arguments is 2 * 10^3 plus the sign, or –2000.

The result of the addition (c) is also returned in those three parts.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
STR$_TRU	String truncation warning. The fixed-length destination string could not contain all the characters.

CONDITION VALUES SIGNALED

LIB$_INVARG	Invalid argument.
STR$_FATINTERR	Fatal internal error. An internal consistency check has failed. This usually indicates an internal error in the Run-Time Library and should be reported to DIGITAL in a Software Performance Report (SPR).
STR$_ILLSTRCLA	Illegal string class. The class code found in the class field of a descriptor is not a string class code allowed by the VAX Procedure Calling and Condition Handling Standard.
STR$_INSVIRMEM	Insufficient virtual memory. STR$ADD could not allocate heap storage for a dynamic or temporary string.
STR$_WRONUMARG	Wrong number of arguments.

EXAMPLE

```
100 !+
    ! This is a sample arithmetic program
    ! showing the use of STR$ADD to add
    ! two decimal strings.
    !-
    ASIGN% = 1%
    AEXP% = 3%
    ADIGITS$ = '1'
    BSIGN% = 0%
    BEXP% = -4%
    BDIGITS$ = '2'
    CSIGN% = 0%
    CEXP% = 0%
    CDIGITS$ = '0'
    PRINT "A = "; ASIGN%; AEXP%; ADIGITS$
    PRINT "B = "; BSIGN%; BEXP%; BDIGITS$
    CALL STR$ADD         (ASIGN%, AEXP%, ADIGITS$, &
                          BSIGN%, BEXP%, BDIGITS$,  &
                          CSIGN%, CEXP%, CDIGITS$)
    PRINT "C = "; CSIGN%; CEXP%; CDIGITS$
999 END
```

This BASIC example uses STR$ADD to add two decimal strings, where the following values apply:

A = –1000 (ASIGN = 1, AEXP = 3, ADIGITS = '1')
B = .0002 (BSIGN = 0, BEXP = -4, BDIGITS = '2')

The output generated by this program is listed below; note that the decimal value of C = -999.9998 (CSIGN = 1, CSIGN = –4, CDIGITS = '9999998').

```
A = 1  3 1
B = 0 -4 2
C = 1 -4 9999998
```

STR$ANALYZE_SDESC—Analyze String Descriptor

STR$ANALYZE_SDESC extracts the length and starting address of the data for a variety of string descriptor classes.

FORMAT

STR$ANALYZE_SDESC *inp-dsc ,len ,data-adr*

corresponding jsb entry point

STR$ANALYZE_SDESC_R1

RETURNS

VMS Usage: **cond_value**
type: **longword integer (signed)**
access: **write only**
mechanism: **by value**

ARGUMENTS

inp-dsc

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Input descriptor from which STR$ANALYZE_SDESC extracts the length of the data and the address at which the data starts. The **inp-dsc** argument is the address of a descriptor pointing to the input data.

len

VMS Usage: **word_signed**
type: **word integer (signed)**
access: **write only**
mechanism: **by reference for CALL entry point,**
by value for JSB entry point

Length of the data; this length is extracted from the descriptor by STR$ANALYZE_SDESC. The **len** argument is the address of a signed word integer into which STR$ANALYZE_SDESC writes the data length.

data-adr

VMS Usage: **address**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference for CALL entry point,**
by value for JSB entry point

Address of the data; this address is extracted from the descriptor by STR$ANALYZE_SDESC. The **data-adr** argument is an unsigned longword into which STR$ANALYZE_SDESC writes the address of the data.

DESCRIPTION

STR$ANALYZE_SDESC takes as input a descriptor argument and extracts from the descriptor the length of the data and the address at which the data starts for a variety of string descriptor classes. See LIB$ANALYZE_SDESC for a list of classes.

STR$ANALYZE_SDESC returns the length of the data in the **len** argument and the starting address of the data in the **data-adr** argument.

STR$ANALYZE_SDESC signals an error if an invalid descriptor class is found.

CONDITION VALUES SIGNALED

STR$_ILLSTRCLA	Illegal string class. The class code found in the class field of a descriptor is not a string class code allowed by the VAX Procedure Calling and Condition Handling Standard.

STR$APPEND—Append String

STR$APPEND appends a source string to the end of a destination string. The destination string must be a dynamic or varying string.

FORMAT

STR$APPEND *dst-str ,src-str*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

dst-str

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

Destination string to which STR$APPEND appends the source string. The **dst-str** argument is the address of a descriptor pointing to the destination string. This destination string must be dynamic or varying.

src-str

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Source string that STR$APPEND appends to the end of the destination string. The **src-str** argument is the address of a descriptor pointing to this source string.

CONDITION VALUE RETURNED

SS$_NORMAL	Routine successfully completed.

CONDITION VALUES SIGNALED

STR$_FATINTERR	Fatal internal error. An internal consistency check has failed. This usually indicates an internal error in the Run-Time Library and should be reported to DIGITAL in a Software Performance Report (SPR).
STR$_ILLSTRCLA	Illegal string class. The class code found in the class field of a descriptor is not a string class code allowed by the VAX Procedure Calling and Condition Handling Standard.
STR$_INSVIRMEM	Insufficient virtual memory. STR$APPEND could not allocate heap storage for a dynamic or temporary string.
STR$_STRTOOLON	The combined lengths of the source and destination strings exceeded 65,535.

EXAMPLE

```
10 !+
   ! This example program uses
   ! STR$APPEND to append a source
   ! string to a destination string.
   !-
   DST$ = 'VAX/'
   SRC$ = 'VMS'
   CALL STR$APPEND (DST$, SRC$)
   PRINT "DST$ = ";DST$
   END
```

This BASIC example uses STR$APPEND to append a source string 'VMS', to a destination string 'VAX/'.

The output generated by this program is as follows:

DST$ = VAX/VMS

STR$CASE_BLIND_COMPARE Compare Strings Without Regard to Case

STR$CASE_BLIND_COMPARE compares two input strings of any supported class and data type without regard to whether the alphabetic characters are uppercase or lowercase.

FORMAT

STR$CASE_BLIND_COMPARE *src1-str ,src2-str*

RETURNS

VMS Usage: **cond_value**
type: **longword integer (signed)**
access: **write only**
mechanism: **by value**

The values returned by STR$CASE_BLIND_COMPARE and the conditions to which they translate are as follows:

Returned Value	Condition
-1	**Src1-str** is less than **src2-str**
0	Both are the same (with blank fill for shorter string)
1	**Src1-str** is greater than **src2-str**

ARGUMENTS

src1-str

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

First string. The **src1-str** argument is the address of a descriptor pointing to the first string.

src2-str

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Second string. The **src2-str** argument is the address of a descriptor pointing to the second string.

DESCRIPTION

STR$CASE_BLIND_COMPARE does not distinguish between uppercase and lowercase characters. The contents of both strings are converted to uppercase before the strings are compared, but the source strings themselves are not changed. STR$CASE_BLIND_COMPARE uses the DEC Multinational Character Set.

CONDITION VALUE SIGNALED

STR$_ILLSTRCLA — Illegal string class. The class code found in the class field of a descriptor is not a string class code allowed by the VAX Procedure Calling and Condition Handling Standard.

EXAMPLE

```
PROGRAM CASE_BLIND(INPUT, OUTPUT);
{+}
{  This program demonstrates the use of
{  STR$CASE_BLIND_COMPARE.
{
{  First, declare the external function.
{-}
FUNCTION STR$CASE_BLIND_COMPARE(STR1 : VARYING
        [A] OF CHAR; STR2 : VARYING [B] OF
        CHAR) : INTEGER; EXTERN;
{+}
{  Declare the variables to be used in the
{  main program.
{-}
VAR
  STRING1       : VARYING [256] OF CHAR;
  STRING2       : VARYING [256] OF CHAR;
  RET_STATUS    : INTEGER;
{+}
{  Begin the main program.  Read values for
{  the strings to be compared.  Call
{  STR$CASE_BLIND_COMPARE.  Print the
{  result.
{-}
BEGIN
  WRITELN('ENTER THE FIRST STRING: ');
  READLN(STRING1);
  WRITELN('ENTER THE SECOND STRING: ');
  READLN(STRING2);
  RET_STATUS := STR$CASE_BLIND_COMPARE(STRING1, STRING2);
  WRITELN(RET_STATUS);
END.
```

This PASCAL example shows how to call STR$CASE_BLIND_COMPARE to determine whether two strings are equal regardless of case. One example of the output of this program is as follows:

```
$ RUN CASE_BLIND
ENTER THE FIRST STRING:  KITTEN
ENTER THE SECOND STRING:  kItTeN
     0
```

STR$COMPARE—Compare Two Strings

STR$COMPARE compares the contents of two strings. If the strings are unequal in length, the shorter string is considered to be filled with blanks to the length of the longer string before the comparison is made.

FORMAT

STR$COMPARE *src1-str ,src2-str*

RETURNS

VMS Usage: **cond_value**
type: **longword integer (signed)**
access: **write only**
mechanism: **by value**

The values returned by STR$COMPARE and the conditions to which they translate are as follows:

Returned Value	Condition
−1	**Src1-str** is less than **src2-str**
0	**Src1-str** is equal to **src2-str**
1	**Src1-str** is greater than **src2-str**

ARGUMENTS

src1-str

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

First string. The **src1-str** argument is the address of a descriptor pointing to the first string.

src2-str

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Second string. The **src2-str** argument is the address of a descriptor pointing to the second string.

CONDITION VALUE SIGNALED

STR$_ILLSTRCLA	Illegal string class. The class code found in the class field of a descriptor is not a string class code allowed by the VAX Procedure Calling and Condition Handling Standard.

EXAMPLE

```
100 EXTERNAL INTEGER FUNCTION STR$COMPARE
    SRC1$ = 'ABC'
    SRC2$ = 'BCD     '
    !+
    !  Note that STR$COMPARE will treat SRC1$ as if it were the same
    !  length as SRC2$ for the purpose of the comparison.  Thus, it
    !  will treat the contents of SRC1$ as 'ABC     '.  However, it
    !  will only 'treat' the contents as longer; the contents of
    !  the source string are not actually changed.
    !-
    I% = STR$COMPARE(SRC1$, SRC2$)
    IF I% = 1 THEN RESULT$ = ' IS GREATER THAN '
    IF I% = 0 THEN RESULT$ = ' IS EQUAL TO '
    IF I% = -1 THEN RESULT$ = ' IS LESS THAN '
    PRINT SRC1$; RESULT$; SRC2$
999 END
```

This BASIC program uses STR$COMPARE to compare two strings. The output generated by this program is as follows:

ABC IS LESS THAN BCD

STR$COMPARE_EQL—Compare Two Strings for Equality

STR$COMPARE_EQL compares two strings to see if they have the same length and contents. Uppercase and lowercase characters are not considered equal.

FORMAT

STR$COMPARE_EQL *src1-str ,src2-str*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

The values returned by STR$COMPARE and the conditions to which they translate are as follows:

Returned Value	Condition
0	The length and the contents of **src1-str** are equal to the length and contents of **src2-str**.
1	Either the length of **src1-str** is not equal to the length of **src2-str**, or the contents of **src1-str** are not equal to the contents of **src2-str**, or both.

ARGUMENTS

src1-str

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

First source string. The **src1-str** argument is the address of a descriptor pointing to the first source string.

src2-str

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Second source string. The **src2-str** argument is the address of a descriptor pointing to the second source string.

CONDITION VALUE SIGNALED

STR$_ILLSTRCLA	Illegal string class. The class code found in the class field of a descriptor is not a string class code allowed by the VAX Procedure Calling and Condition Handling Standard.

EXAMPLE

```
PROGRAM COMPARE_EQL(INPUT, OUTPUT);
{+}
{  This program demonstrates the use of
{  STR$COMPARE_EQL to compare two strings.
{  Strings are considered equal only if they
{  have the same contents and the same length.
{
{  First, declare the external function.
{-}
FUNCTION STR$COMPARE_EQL(SRC1STR : VARYING
         [A] OF CHAR; SRC2STR : VARYING [B]
         OF CHAR) : INTEGER; EXTERN;
{+}
{  Declare the variables used in the main program.
{-}
VAR
  STRING1       : VARYING [256] OF CHAR;
  STRING2       : VARYING [256] OF CHAR;
  RET_STATUS    : INTEGER;
{+}
{  Begin the main program.  Read the strings
{  to be compared.  Call STR$COMARE_EQL to compare
{  the strings.  Print the result.
{-}
BEGIN
  WRITELN('ENTER THE FIRST STRING: ');
  READLN(STRING1);
  WRITELN('ENTER THE SECOND STRING: ');
  READLN(STRING2);
  RET_STATUS := STR$COMPARE_EQL(STRING1, STRING2);
  WRITELN(RET_STATUS);
END.
```

This PASCAL example demonstrates the use of STR$COMPARE_EQL. A sample of the output generated by this program is as follows:

```
$ RUN COMPARE_EQL
ENTER THE FIRST STRING:  frog
ENTER THE SECOND STRING:  Frogs
     1
```

STR$COMPARE_MULTI—Compare Two Strings for Equality Using Multinational Character Set

STR$COMPARE_MULTI compares two character strings for equality using the DEC Multinational character set.

FORMAT

STR$COMPARE_MULTI *src1-str ,src2-str [,case-blind-flag] [,foreign-lang]*

RETURNS

VMS Usage: **cond_value**
type: **longword integer (signed)**
access: **write only**
mechanism: **by value**

The values returned by STR$COMPARE_MULTI and the conditions to which they translate are as follows:

Returned Value	Condition
-1	**Src1-str** is less than **src2-str**.
0	Both strings are the same; the shorter string is blank filled.
1	**Src1-str** is greater than **src2-str**.

ARGUMENTS

src1-str

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**
First string in the comparison. The **src1-str** argument is the address of a descriptor pointing to the first string.

src2-str

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**
Second string in the comparison. The **src2-str** argument is the address of a descriptor pointing to the second string.

case-blind-flag

VMS Usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

A single flag bit. The **case-blind-flag** argument is a signed longword integer that contains this flag bit. The default value of **case-blind-flag** is zero.

Bit	Symbol	Meaning
0	CASEBLIND	If set, uppercase and lowercase characters are equivalent.

foreign-lang

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Indicator which determines the foreign language table to be used. The **foreign-lang** argument is an unsigned longword that contains this foreign language table indicator. The default value of **foreign-lang** is 1.

Value	Language
1	Multinational table
2	Danish table
3	Finnish/Swedish table
4	German table
5	Norwegian table
6	Spanish table

DESCRIPTION

STR$COMPARE_MULTI compares two character strings to see if they have the same contents. Two strings are "equal" if they contain the same characters in the same sequence, even if one of them is blank filled to a longer length than the other. The DEC Multinational character set, or foreign language variations of the DEC Multinational character set, is used in the comparison.

See the *VAX/VMS I/O Reference Volume* for more information about the DEC Multinational character set.

CONDITION VALUES SIGNALED

STR$_ILLSTRCLA	Illegal string class. Severe error. The descriptor of **src1-str** and/or **src2-str** contains an class code that is not supported by the VAX Procedure Calling and Condition Handling Standard.
LIB$_INVARG	Invalid Argument. Severe error.

STR$CONCAT—Concatenate Two or More Strings

STR$CONCAT takes up to 254 source strings and concatenates them into a single destination string.

FORMAT

STR$CONCAT *dst-str ,src1-str [... ,srcn-str]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

dst-str

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

Destination string into which STR$CONCAT concatenates all specified source strings. The **dst-str** argument is the address of a descriptor pointing to this destination string.

src1-str

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

First source string. The **src1-str** argument is the address of a descriptor pointing to the first source string. STR$CONCAT requires at least one source string.

srcn-str

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Nth source string. The **scrn-str** argument is the address of a descriptor pointing to the Nth source string. The largest value of N that STR$CONCAT allows is 254.

DESCRIPTION

STR$CONCAT concatenates all specified source strings into a single destination string. The strings can be of any class and data type, provided that the length fields of the descriptors indicate the strings' lengths in bytes.

A warning status is returned if one or more input characters were not copied to the destination string.

You must specify at least one source string, and you may specify up to 254 source strings. The maximum length of the concatenated string is 65,535 bytes.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed. All characters in the input strings were copied into the destination string.
STR$_TRU	String truncation warning. One or more input characters were not copied into the destination string. This can happen when the destination is a fixed-length string.

CONDITION VALUES SIGNALED

STR$_FATINTERR	Fatal internal error. An internal consistency check has failed. This usually indicates an internal error in the Run-Time Library and should be reported to DIGITAL in a Software Performance Report (SPR).
STR$_ILLSTRCLA	Illegal string class. The class code found in the class field of a descriptor is not a string class code allowed by the VAX Procedure Calling and Condition Handling Standard.
STR$_INSVIRMEM	Insufficient virtual memory. STR$CONCAT could not allocate heap storage for a dynamic or temporary string.
STR$_STRTOOLON	String length exceeds 65,535 bytes.
STR$_WRONUMARG	Wrong number of arguments. You tried to pass fewer than 2 or more than 255 arguments to STR$CONCAT.

EXAMPLES

1

```
10 !+
   !  This example program uses STR$CONCAT
   !  to concatenate four source strings into a
   !  single destination string.
   !-
   EXTERNAL INTEGER FUNCTION STR$CONCAT
   STATUS% = STR$CONCAT (X$, 'A', 'B', 'C', 'D')
   PRINT "X$ = ";X$
   END
```

The output generated by this BASIC program is as follows:

X$ = ABCD

2

```
MSG1:                  .ASCII /David Foster /      ; first string
MSG1_LEN = .-MSG1                                  ; its length
MSG2:                  .ASCII /wrote this book./   ; second string
MSG2_LEN = .-MSG2                                  ; its length
RESULT:                .BLKB MSG1_LEN + MSG2_LEN   ; string to hold concatenation
MSG1_DSC:              .WORD MSG1_LEN              ; DSC$W_LENGTH
                       .BYTE 14                    ; DSC$B_DTYPE
                       .BYTE 1                     ; DSC$B_CLASS
                       .ADDRESS MSG1               ; DSC$A_POINTER
MSG2_DSC:              .WORD MSG2_LEN              ; DSC$W_LENGTH
                       .BYTE 14                    ; DSC$B_DTYPE
                       .BYTE 1                     ; DSC$B_CLASS
                       .ADDRESS MSG2               ; DSC$A_POINTER
RESULT_DSC:            .WORD MSG1_LEN + MSG2_LEN   ; DSC$W_LENGTH
                       .BYTE 14                    ; DSC$B_DTYPE
                       .BYTE 1                     ; DSC$B_CLASS
                       .ADDRESS RESULT             ; DSC$A_POINTER
        .ENTRY  EXAM1, ^M<>      ; entry point
        PUSHAQ  MSG2_DSC         ; Push the descriptors
        PUSHAQ  MSG1_DSC         ; in reverse order
        PUSHAQ  RESULT_DSC       ; .
        CALLS   #3, G^STR$CONCAT         ; concatenate strings
        PUSHAQ  RESULT_DSC               ; descr. of string to display
        CALLS   #1, G^LIB$PUT_OUTPUT     ; display it
        RET                              ; return to calling routine
        .END    EXAM1
```

The output generated by this MACRO program is as follows:

David Foster wrote this book.

STR$COPY_DX—Copy a Source String Passed by Descriptor to a Destination String

STR$COPY_DX copies a source string to a destination string. Both strings are passed by descriptor.

FORMAT

STR$COPY_DX *dst-str ,src-str*

corresponding jsb entry point

STR$COPY_DX_R8

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

dst-str

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

Destination string into which STR$COPY_DX writes the source string. The **dst-str** argument is the address of a descriptor pointing to the destination string.

The class field determines how the copy operation is handled. For further information, see the Description section.

src-str

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Source string that STR$COPY_DX copies into the destination string. The **src-str** argument is the address of a descriptor pointing to this source string. The descriptor class of the source string can be unspecified, fixed length, dynamic, scalar decimal, array, noncontiguous array, or varying.

(See the description of LIB$ANALYZE_SDESC for possible restrictions.)

DESCRIPTION

STR$COPY_DX copies a source string to a destination string, where both strings are passed by descriptor. All conditions except truncation are signaled; truncation is returned as a warning condition value (bit 0 is clear) in R0.

STR$COPY_DX passes the source string by descriptor. In addition, an equivalent JSB entry point is provided, with R0 being the first argument (the descriptor of the destination string), and R1 the second (the descriptor of the source string).

Depending on the class of the destination string, the following actions occur:

Class Field	Action
DSC$K_CLASS_S,Z,SD,A,NCA	Copy the source string. If needed, space is filled or truncated on the right.
DSC$K_CLASS_D	If the area specified by the destination descriptor is large enough to contain the source string, copy the source string and set the new length in the destination descriptor. If the area specified is not large enough, return the previous space allocation (if any) and then dynamically allocate the amount of space needed. Copy the source string and set the new length and address in the destination descriptor.
DSC$K_CLASS_VS	Copy source string to destination string up to the limit of DSC$W_MAXSTRLEN with no padding. Readjust current length field to actual number of bytes copied.

CONDITION VALUES RETURNED

SS$_NORMAL — Procedure successfully completed. All characters in the input string were copied to the destination string.

STR$_TRU — String truncation warning. The fixed-length destination string could not contain all of the characters copied from the source string.

CONDITION VALUES SIGNALED

STR$_FATINTERR — Fatal internal error. An internal consistency check has failed. This usually indicates an internal error in the Run-Time Library and should be reported to DIGITAL in a Software Performance Report (SPR).

STR$_ILLSTRCLA — Illegal string class. The class code found in the class field of a descriptor is not a string class code allowed by the VAX Procedure Calling and Condition Handling Standard.

STR$_INSVIRMEM — Insufficient virtual memory. STR$COPY_DX could not allocate heap storage for a dynamic or temporary string.

STR$COPY_R—Copy Source String Passed by Reference to Destination String

STR$COPY_R copies a source string passed by reference to a destination string.

FORMAT

STR$COPY_R *dst-str ,src-len ,src-str*

corresponding jsb entry point

STR$COPY_R_R8

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

dst-str

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

Destination string into which STR$COPY_R copies the source string. The **dst-str** argument is the address of a descriptor pointing to the destination string.

The class field determines the appropriate action.

src-len

VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **read only**
mechanism: **by reference**

Length of the source string. The **src-len** argument is the address of an unsigned word containing the length of the source string.

src-str

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by reference**

Source string which STR$COPY_R copies into the destination string. The **src-str** argument is the address of the source string.

See the description of LIB$ANALYZE_SDESC for possible restrictions.

DESCRIPTION

STR$COPY_R copies a source string passed by reference to a destination string. All conditions except truncation are signaled; truncation is returned as a warning condition value (bit 0 clear) in R0.

A JSB entry point is provided, with R0 being the first argument, R1 the second, and R2 the third. The length argument is passed in bits 15:0 of R1.

Depending on the class of the destination string, the following actions occur:

Class Field	Action
DSC$K_CLASS_S,Z,SD,A,NCA	Copy the source string. If needed, space is filled or truncated on the right.
DSC$K_CLASS_D	If the area specified by the destination descriptor is large enough to contain the source string, copy the source string and set the new length in the destination descriptor. If the area specified is not large enough, return the previous space allocation (if any) and then dynamically allocate the amount of space needed. Copy the source string and set the new length and address in the destination descriptor.
DSC$K_CLASS_VS	Copy source string to destination string up to the limit of DSC$W_MAXSTRLEN with no padding. Readjust current length field to actual number of bytes copied.

CONDITION VALUES RETURNED

SS$_NORMAL	Procedure successfully completed. All characters in the input string were copied to the destination string.
STR$_TRU	String truncation warning. The fixed-length destination string could not contain all of the characters copied from the source string.

CONDITION VALUES SIGNALED

STR$_FATINTERR	Fatal internal error. An internal consistency check has failed. This usually indicates an internal error in the Run-Time Library and should be reported to DIGITAL in a Software Performance Report (SPR).
STR$_ILLSTRCLA	Illegal string class. The class code found in the class field of a descriptor is not a string class code allowed by the VAX Procedure Calling and Condition Handling Standard.
STR$_INSVIRMEM	Insufficient virtual memory. STR$COPY_R could not allocate heap storage for a dynamic or temporary string.

STR$DIVIDE—Divide Two Decimal Strings

STR$DIVIDE divides two decimal strings.

FORMAT

STR$DIVIDE *asign ,aexp ,adigits ,bsign ,bexp ,bdigits ,tot-digits ,rnd-trunc ,csign ,cexp ,cdigits*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

asign

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Sign of the first operand. The **asign** argument is the address of an unsigned longword containing the first operand's sign. Zero is considered positive; 1 is considered negative.

aexp

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Power of 10 by which **adigits** has to be multiplied to get the absolute value of the first operand. The **aexp** argument is the address of the first operand's exponent.

adigits

VMS Usage: **char_string**
type: **num. string, unsigned**
access: **read only**
mechanism: **by descriptor**

First operand's numeric string. The **adigits** argument is the address of a descriptor pointing to the first operand's numeric string. The string must be an unsigned decimal number.

bsign

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Sign of the second operand. The **bsign** argument is the address of an unsigned longword containing the second operand's string. Zero is considered positive; 1 is considered negative.

bexp

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Power of 10 by which **bdigits** has to be multiplied to get the absolute value of the second operand. The **bexp** argument is the address of the second operand's exponent.

bdigits

VMS Usage: **char_string**
type: **num. string, unsigned**
access: **read only**
mechanism: **by descriptor**

Second operand's numeric string. The **bdigits** argument is the address of a descriptor pointing to the second operand's number string. The string must be an unsigned decimal number.

tot-digits

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Number of digits to the right of the decimal point. The **tot-digits** argument is the address of s signed longword integer containing the number of total digits. STR$DIVIDE uses this number to carry out the division.

rnd-trunc

VMS Usage: **longword_unsigned**
type: **aligned bit string**
access: **read only**
mechanism: **by reference**

Indicator of whether STR$DIVIDE is to round or truncate the result; zero means truncate; 1 means round. The **rnd-trunc** argument is the address of an aligned bit string containing this indicator.

csign

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **write only**
mechanism: **by reference**

Sign of the result. The **csign** argument is the address of a signed longword integer containing the sign of the result. Zero is considered positive; 1 is considered negative.

cexp

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **write only**
mechanism: **by reference**

Power of 10 by which **cdigits** has to be multiplied to get the absolute value of the result. The **cexp** argument is the address of a signed longword integer containing the exponent.

cdigits
VMS Usage: **char_string**
type: **num. string, unsigned**
access: **write only**
mechanism: **by descriptor**
Result's numeric string. The **cdigits** argument is the address of a descriptor pointing to the numeric string of the result. This string is an unsigned decimal number.

DESCRIPTION

STR$DIVIDE divides two decimal strings. The divisor and dividend are passed to STR$DIVIDE in three parts: (1) the numeric string, (2) the power of 10 needed to obtain the absolute value, and (3) the sign of the decimal number. The result of the division is also returned in those three parts.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
STR$_TRU	String truncation warning. The fixed-length destination string could not contain all of the characters.

CONDITION VALUES SIGNALED

LIB$_INVARG	Invalid argument.
STR$_DIVBY_ZER	Division by zero.
STR$_FATINTERR	Fatal internal error. An internal consistency check has failed. This usually indicates an internal error in the Run-Time Library and should be reported to DIGITAL in a Software Performance Report (SPR).
STR$_ILLSTRCLA	Illegal string class. The class code found in the class field of a descriptor is not a string class code allowed by the VAX Procedure Calling and Condition Handling Standard.
STR$_INSVIRMEM	Insufficient virtual memory. STR$DIVIDE could not allocate heap storage for a dynamic or temporary string.
STR$_WRONUMARG	Wrong number of arguments.

EXAMPLE

```
100 !+
    ! This example program uses STR$DIVIDE
    ! to divide two decimal strings, and truncate
    ! the result.
    !-
    ASIGN% = 1%
    AEXP% = 3%
    ADIGITS$ = '1'
    BSIGN% = 0%
    BEXP% = -4%
    BDIGITS$ = '2'
    CSIGN% = 0%
    CEXP% = 0%
    CDIGITS$ = '0'
    PRINT "A = "; ASIGN%; AEXP%; ADIGITS$
```

```
    PRINT "B = "; BSIGN%; BEXP%; BDIGITS$
    CALL STR$DIVIDE      (ASIGN%, AEXP%, ADIGITS$, &
                          BSIGN%, BEXP%, BDIGITS$,  &
                          3%, 0%, CSIGN%, CEXP%, CDIGITS$)
     PRINT "C = "; CSIGN%; CEXP%; CDIGITS$
1500 END
```

This BASIC program uses STR$DIVIDE to divide two decimal strings, A divided by B, where the following values apply:

A = –1000 (ASIGN = 1, AEXP = 3, ADIGITS = '1')
B = .0002 (BSIGN = 0, BEXP = –4, BDIGITS = '2')

The output generated by this program is as follows:

```
A = 1  3 1
B = 0 -4 2
C = 1 -3 5000000000
```

Thus, the decimal value of C = –5000000 (CSIGN = 1, CEXP = –3, CDIGITS = 5000000000).

STR$DUPL_CHAR—Duplicate Character n Times

STR$DUPL_CHAR generates a string containing n duplicates of the input character. If the destination string is an "empty" dynamic string descriptor, STR$DUPL_CHAR will allocate and initialize the string.

FORMAT

STR$DUPL_CHAR *dst-str [,length] [,char]*

corresponding jsb entry point

STR$DUPL_CHAR_R8

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

dst-str

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

Destination string into which STR$DUPL_CHAR writes **length** copies of the input character. The **dst-str** argument is the address of a descriptor pointing to the destination string.

length

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Number of times **char** will be duplicated. The **length** argument is the address of a signed longword integer containing the number. This is an optional argument. If omitted, the default is 1.

char

VMS Usage: **byte_unsigned**
type: **byte (unsigned)**
access: **read only**
mechanism: **by reference**

ASCII character which STR$DUPL_CHAR writes **length** times into the destination string. The **char** argument is the address of an unsigned byte containing this character. This is an optional argument. If omitted, the default is a space.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
STR$_NEGSTRLEN	Routine successfully completed. The length argument contained a negative value; zero was used.
STR$_TRU	String truncation warning. The fixed-length destination string could not contain all of the characters.

CONDITION VALUES SIGNALED

STR$_FATINTERR	Fatal internal error. An internal consistency check has failed. This usually indicates an internal error in the Run-Time Library and should be reported to DIGITAL in a Software Performance Report (SPR).
STR$_ILLSTRCLA	Illegal string class. The class code found in the class field of a descriptor is not a string class code allowed by the VAX Procedure Calling and Condition Handling Standard.
STR$_INSVIRMEM	Insufficient virtual memory. STR$DUPL_CHAR could not allocate heap storage for a dynamic or temporary string.
STR$_STRTOOLON	String length exceeds 65,535 bytes.

EXAMPLE

```
10  !+
    ! This example uses STR$DUPL_CHAR to
    ! duplicate the character 'A' four times.
    !-
    EXTERNAL INTEGER FUNCTION STR$DUPL_CHAR
    STATUS% = STR$DUPL_CHAR (X$, 4%, 'A' BY REF)
    PRINT X$
    END
```

These BASIC statements set X$ equal to 'AAAA'.

STR$FIND_FIRST_IN_SET—Find First Character in a Set of Characters

STR$FIND_FIRST_IN_SET searches a string one character at a time, from left to right, comparing each character in the string to every character in a specified set of characters for which it is searching. STR$FIND_FIRST_IN_SET returns the position in the string where the first matching character was found. Zero is returned if no match is found.

FORMAT

STR$FIND_FIRST_IN_SET *src-str ,set-of-chars*

RETURNS

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **write only**
mechanism: **by value**

Position in **src-str** where the first match is found; zero if no match is found.

ARGUMENTS

src-str

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

String which STR$FIND_FIRST_IN_SET compares to the set of characters, looking for the first match. The **src-str** argument is the address of a descriptor pointing to the character string.

set-of-chars

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Set of characters which STR$FIND_FIRST_IN_SET is searching for in the string. The **src-str** argument is the address of a descriptor pointing to the set of characters.

DESCRIPTION

STR$FIND_FIRST_IN_SET compares each character in the string to every character in the specified set of characters. As soon as the first match is found, STR$FIND_FIRST_IN_SET returns the position in the string where the matching character was found. If no match is found, 0 is returned. If either **src-str** or **set-of-chars** is of zero length, 0 is returned.

CONDITION VALUE SIGNALED

STR$_ILLSTRCLA	Illegal string class. The class code found in the class field of a descriptor is not a string class code allowed by the VAX Procedure Calling and Condition Handling Standard.

EXAMPLE

```
PROGRAM FIND_FIRST(INPUT, OUTPUT);
{+}
{  This example uses STR$FIND_FIRST_IN_SET
{  to find the first character in the source
{  string (STRING1) which matches a character
{  in the set of characters being searched for
{  (CHARS).
{
{  First, declare the external function.
{-}
FUNCTION STR$FIND_FIRST_IN_SET(STRING :
          VARYING [A] OF CHAR; SETOFCHARS :
          VARYING [B] OF CHAR) : INTEGER;
          EXTERN;
{+}
{  Declare the variables used in the main program.
{-}
VAR
  STRING1       : VARYING [256] OF CHAR;
  CHARS         : VARYING [256] OF CHAR;
  RET_STATUS    : INTEGER;
{+}
{  Begin the main program.  Read the source string
{  and the set of characters being searched for.  Call
{  STR$FIND_FIRST_IN_SET to find the first match.
{  Print the result.
{-}
BEGIN
  WRITELN('ENTER THE STRING: ');
  READLN(STRING1);
  WRITELN('ENTER THE SET OF CHARACTERS: ');
  READLN(CHARS);
  RET_STATUS := STR$FIND_FIRST_IN_SET(STRING1, CHARS);
  WRITELN(RET_STATUS);
END.
```

This PASCAL program demonstrates the use of STR$FIND_FIRST_IN_SET. If you run this program and set STRING1 equal to ABCDEFGHIJK and CHARS equal to XYZA, the value of RET_STATUS will be 1.

STR$FIND_FIRST_NOT_IN_SET Find First Character that Does Not Occur in Set

STR$FIND_FIRST_NOT_IN_SET searches a string, comparing each character to the characters in a specified set of characters. The string is searched character by character, from left to right. STR$FIND_FIRST_NOT_IN_SET returns the position of the first character in the string that does not match any of the characters in the selected set of characters.

FORMAT

STR$FIND_FIRST_NOT_IN_SET *src-str ,set-of-chars*

RETURNS

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **write only**
mechanism: **by value**

Position in **src-str** where a nonmatch was found.

Returned value	Condition
0	Either all characters in **src-str** match some character in **set-of-chars**, or there were no characters in **set-of-chars**.
1	Either the first nonmatching character in **src-str** was found in position 1, or there were no characters in **src-str**.
N	The first nonmatching character was found in position N within **src-str**.

ARGUMENTS

src-str

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

String which STR$FIND_FIRST_NOT_IN_SET searches. The **src-str** argument is the address of a descriptor pointing to the string.

set-of-chars

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

The set of characters which STR$FIND_FIRST_NOT_IN_SET compares to the string, looking for a nonmatch. The **set-of-chars** argument is the address of a descriptor pointing to this set of characters.

DESCRIPTION

STR$FIND_FIRST_NOT_IN_SET searches **src-str** one character at a time, from left to right, comparing each character in the string to every character in **set-of-chars**. When STR$FIND_FIRST_NOT_IN_SET finds a character from the string that is not in **set-of-chars**, it stops searching and returns, as the value of STR$FIND_FIRST_NOT_IN_SET, the position in **src-str** where it found the nonmatching character. If all characters in the string match some character in the set of characters, STR$FIND_FIRST_NOT_IN_SET returns 0. If the string is of zero length, the position returned is 1 since none of the elements in the set of characters (particularly the first element) will be found in the string. If there are no characters in the set of characters, zero is returned since "nothing" can always be found.

CONDITION VALUE SIGNALED

STR$_ILLSTRCLA	Illegal string class. The class code found in the class field of a descriptor is not a string class code allowed by the VAX Procedure Calling and Condition Handling Standard.

EXAMPLE

```
PROGRAM NOT_IN_SET(INPUT, OUTPUT);
{+}
{  This example uses STR$FIND_FIRST_NOT_IN_SET
{  to find the position of the first nonmatching
{  character from a set of characters (CHARS)
{  in a source string (STRING1).
{
{  First, declare the external function.
{-}
FUNCTION STR$FIND_FIRST_NOT_IN_SET(STRING :
          VARYING [A] OF CHAR; SETOFCHARS :
          VARYING [B] OF CHAR) : INTEGER;
          EXTERN;
{+}
{  Declare the variables used in the main program.
{-}
VAR
  STRING1        : VARYING [256] OF CHAR;
  CHARS          : VARYING [256] OF CHAR;
  RET_STATUS     : INTEGER;
{+}
{  Begin the main program.  Read the source string
{  and set of characters.  Call STR$FIND_FIRST_NOT_IN_SET.
{  Print the result.
{-}
BEGIN
  WRITELN('ENTER THE STRING: ');
  READLN(STRING1);
  WRITELN('ENTER THE SET OF CHARACTERS: ');
  READLN(CHARS);
  RET_STATUS := STR$FIND_FIRST_NOT_IN_SET(STRING1, CHARS);
  WRITELN(RET_STATUS);
END.
```

This PASCAL program demonstrates the use of STR$FIND_FIRST_NOT_IN_SET. If you run this program and set STRING1 equal to FORTUNATE and CHARS equal to FORT, the value of RET_STATUS will be 5.

STR$FIND_FIRST_SUBSTRING—Find First Substring in Input String

STR$FIND_FIRST_SUBSTRING finds the first substring (in a provided list of substrings) occurring in a given string.

FORMAT

STR$FIND_FIRST_SUBSTRING
src-str ,index ,sub-string-index
,sub-string1 ...
[,sub-stringn]

RETURNS

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

The values returned by STR$FIND_FIRST_SUBSTRING and the conditions to which they translate are as follows:

Returned Value	Condition
0	**Src-str** did not contain any of the specified substrings.
1	STR$FIND_FIRST_SUBSTRING found at least one of the specified substrings in the string.

ARGUMENTS

src-str

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

String that STR$FIND_FIRST_SUBSTRING searches. The **src-str** argument is the address of a descriptor pointing to the string.

index

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **write only**
mechanism: **by reference**

Earliest position within **src-str** at which STR$FIND_FIRST_SUBSTRING found a matching substring; zero if no matching substring was found. The **index** argument is the address of a signed longword integer containing this position.

sub-string-index

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **write only**
mechanism: **by reference**

Ordinal number of the **sub-string** that matched (1 for the first, 2 for the second, and so on), or zero if STR$FIND_FIRST_SUBSTRING found no substrings that matched. The **sub-string-index** argument is the address of a signed longword integer containing this ordinal number.

sub-string1

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

First specified substring for which STR$FIND_FIRST_SUBSTRING searches in **src-str**. The **src-str** argument is the address of a descriptor pointing to the first substring.

sub-stringn

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Nth specified substring which STR$FIND_FIRST_SUBSTRING will search for in the string. The **sub-stringn** argument is the address of a descriptor pointing to the Nth substring.

DESCRIPTION

STR$FIND_FIRST_SUBSTRING takes as input a string to be searched and an unspecified number of substrings for which to search. It searches the specified string and returns the position of the substring which is found earliest in the string. This is not necessarily the position of the first substring specified. (See the example at the end of this routine description.)

Unlike many of the compare and search routines, STR$FIND_FIRST_SUBSTRING does not return the position in a return value. The position of the substring which is found earliest in the string is returned in the **index** argument. If none of the specified substrings are found in the string, the value of **index** is zero.

Zero length strings, or 'null' arguments, produce unexpected results. Any time the procedure is called with a null substring as an argument, STR$FIND_FIRST_SUBSTRING will always return the position of the null substring as the first substring found. All other substrings will be interpreted as appearing in the string after the null string.

CONDITION VALUES SIGNALED

STR$_ILLSTRCLA	Illegal string class. The class code found in the class field of a descriptor is not a string class code allowed by the VAX Procedure Calling and Condition Handling Standard.
STR$_WRONUMARG	Wrong number of arguments. You must supply at least one substring.

EXAMPLE

```
1   !+
    ! This is a BASIC program demonstrating the use of
    ! STR$FIND_FIRST_SUBSTRING.  This program takes as input
    ! four strings which are listed in a data statement
    ! at the end of the program.  STR$FIND_FIRST_SUBSTRING
    ! is called four times (once for each string)
    ! to find the first substring occuring in the given
    ! string.
    !-
    OPTION TYPE = EXPLICIT
    DECLARE STRING      MATCH_STRING
    DECLARE LONG        RET_STATUS, &
                        INDEX, &
                        I, &
                        SUB_STRING_NUM
     EXTERNAL LONG FUNCTION STR$FIND_FIRST_SUBSTRING
     FOR I = 1 TO 4
        READ MATCH_STRING
        RET_STATUS = STR$FIND_FIRST_SUBSTRING( MATCH_STRING, &
            INDEX, SUB_STRING_NUM, 'ING', 'TH', 'CK')
        IF RET_STATUS = 0% THEN
            PRINT MATCH_STRING;" did not contain any of the substrings"
        ELSE
            SELECT SUB_STRING_NUM
                CASE 1
                    PRINT MATCH_STRING;" contains ING at position";INDEX
                CASE 2
                    PRINT MATCH_STRING;" contains TH at position";INDEX
                CASE 3
                    PRINT MATCH_STRING;" contains CK at position";INDEX
            END SELECT
        END IF
     NEXT I
2    DATA CHUCKLE, RAINING, FOURTH, THICK
3    END
```

This BASIC program demonstrates the use of STR$FIND_FIRST_SUBSTRING. The output generated by this program is as follows:

```
$ BASIC FINDSUB
$ LINK FINDSUB
$ RUN FINDSUB
CHUCKLE contains CK at position 4
RAINING contains ING at position 5
FOURTH contains TH at position 5
THICK contains TH at position 1
```

Note that "THICK" contains both the substrings "TH" and "CK". However, since "TH" occurs earlier in the string than "CK", its ordinal number is returned in **substring-index**, and the point at which "TH" occurs is returned in **index**.

STR$FREE1_DX—Free One Dynamic String

STR$FREE1_DX deallocates one dynamic string.

FORMAT

STR$FREE1_DX *dsc-adr*

corresponding jsb entry point

STR$FREE1_DX_R4

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENT

dsc-adr

VMS Usage:	**char_string**
type:	**character string (unsigned)**
access:	**modify**
mechanism:	**by descriptor**

Dynamic string descriptor of the dynamic string which STR$FREE1_DX deallocates. The **dsc-adr** argument is the address of a descriptor pointing to the string to be deallocated. The class field (DSC$B_CLASS) is checked.

DESCRIPTION

STR$FREE1_DX deallocates the described string space and flags the descriptor as describing no string at all (DSC$A_POINTER = 0, DSC$W_LENGTH = 0).

CONDITION VALUES RETURNED

SS$_NORMAL	Procedure successfully completed.

CONDITION VALUES SIGNALED

STR$_FATINTERR	Fatal internal error. An internal consistency check has failed. This usually indicates an internal error in the Run-Time Library and should be reported to DIGITAL in a Software Performance Report (SPR).
STR$_ILLSTRCLA	Illegal string class. The class code found in the class field of a descriptor is not a string class code allowed by the VAX Procedure Calling and Condition Handling Standard.

STR$GET1_DX—Allocate One Dynamic String

STR$GET1_DX allocates a specified number of bytes of dynamic virtual memory to a specified dynamic string descriptor.

FORMAT

STR$GET1_DX *len ,str*

corresponding jsb entry point

STR$GET1_DX_R4

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

len

VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **read only**
mechanism: **by reference**

Number of bytes which STR$GET1_DX allocates. The **len** argument is the address of an unsigned word containing this number.

str

VMS Usage: **char_string**
type: **character string**
access: **modify**
mechanism: **by descriptor**

Dynamic string descriptor to which STR$GET1_DX allocates the area. The **str** argument is the address of an unsigned quadword containing the string descriptor.

The class field (DSC$B_CLASS) is checked.

DESCRIPTION

STR$GET1_DX allocates a specified number of bytes of dynamic virtual memory to a specified string descriptor. The descriptor must be dynamic.

If the string descriptor already has dynamic memory allocated to it, but the amount allocated is less than **len**, STR$GET1_DX deallocates that space before it allocates new space.

STR$GET1_DX is the only recommended method for allocating a dynamic descriptor. Simply filling in the length and pointer fields of a dynamic string descriptor can cause serious and unexpected problems with string management.

To deallocate dynamic strings, call STR$FREE1_DX.

CONDITION VALUE RETURNED

SS$_NORMAL	Procedure successfully completed.

CONDITION VALUES SIGNALED

STR$_FATINTERR	Fatal internal error. An internal consistency check has failed. This usually indicates an internal error in the Run-Time Library and should be reported to DIGITAL in a Software Performance Report (SPR).
STR$_ILLSTRCLA	Illegal string class. The class code found in the class field of a descriptor is not a string class code allowed by the VAX Procedure Calling and Condition Handling Standard.
STR$_INSVIRMEM	Insufficient virtual memory. STR$GET1_DX could not allocate heap storage for a dynamic or temporary string.

STR$LEFT—Extract a Substring of a String

STR$LEFT copies a substring of a source string into a destination string.

FORMAT

STR$LEFT *dst-str ,src-str ,end-pos*

corresponding jsb entry point

STR$LEFT_R8

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

dst-str

VMS Usage:	**char_string**
type:	**character string**
access:	**write only**
mechanism:	**by descriptor**

Destination string into which STR$LEFT copies the substring. The **dst-str** argument is the address of a descriptor pointing to the destination string.

src-str

VMS Usage:	**char_string**
type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

Source string from which STR$LEFT extracts the substring which it copies into the destination string. The **src-str** argument is the address of a descriptor pointing to the source string.

end-pos

VMS Usage:	**longword_signed**
type:	**longword integer (signed)**
access:	**read only**
mechanism:	**by reference**

Relative position in the source string at which the substring ends. The **end-pos** argument is the address of a signed longword integer containing the ending position.

STR$LEFT copies all characters in the source string from position 1 to the position number specified in this **end-pos** argument.

DESCRIPTION

STR$LEFT extracts a substring from a source string and copies that substring into a destination string. STR$LEFT defines the substring by specifying the relative ending position in the source string. The relative starting position in the source string is 1. The source string is unchanged, unless it is also the destination string.

This is a variation of STR$POS_EXTR. Other routines that may be used to extract and copy a substring are STR$RIGHT and STR$LEN_EXTR.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
STR$_ILLSTRPOS	Routine successfully completed, except that an argument referenced a character position outside the specified string. A default value was used.
STR$_ILLSTRSPE	Routine successfully completed, except that the length of the substring was too long for the specified destination string. Default values were used.
STR$_TRU	String truncation warning. The fixed-length destination string could not contain all the characters copied from the source string.

CONDITION VALUES SIGNALED

STR$_FATINTERR	Fatal internal error. An internal consistency check has failed. This usually indicates an internal error in the Run-Time Library and should be reported to DIGITAL in a Software Performance Report (SPR).
STR$_ILLSTRCLA	Illegal string class. The class code found in the class field of a descriptor is not a string class code allowed by the VAX Procedure Calling and Condition Handling Standard.
STR$_INSVIRMEM	Insufficient virtual memory. STR$LEFT could not allocate heap storage for a dynamic or temporary string.

EXAMPLE

```
PROGRAM LEFT(INPUT, OUTPUT);
{+}
{ This PASCAL program demonstrates the use of
{ STR$LEFT.  This program reads in a source string
{ and the ending position of a substring.
{ It returns a substring consisting of all
{ characters from the beginning (left) of the
{ source string to the ending position entered.
{-}
{+}
{ Declare the external procedure, STR$LEFT.
{-}
PROCEDURE STR$LEFT(%DESCR DSTSTR: VARYING
  [A] OF CHAR; SRCSTR  :
  VARYING [B] OF CHAR; ENDPOS :
    INTEGER); EXTERN;
```

```
{+}
{ Declare the variables used by this program.
{-}
VAR
  SRC_STR : VARYING [256] OF CHAR;
  DST_STR : VARYING [256] OF CHAR;
  END_POS : INTEGER;
{+}
{ Begin the main program.  Read the source string
{ and ending position.  Call STR$LEFT.  Print the
{ results.
{-}
BEGIN
  WRITELN('ENTER THE SOURCE STRING: ');
  READLN(SRC_STR);
  WRITELN('ENTER THE ENDING POSITION');
  WRITELN('OF THE SUBSTRING: ');
  READLN(END_POS);
  STR$LEFT(DST_STR, SRC_STR, END_POS);
  WRITELN;
  WRITELN('THE SUBSTRING IS: ',DST_STR);
END.
```

This PASCAL example shows the use of STR$LEFT. The following is one sample of the output of this program:

```
$ PASCAL LEFT
$ LINK LEFT
$ RUN LEFT
ENTER THE SOURCE STRING:  MAGIC CARPET
ENTER THE ENDING POSITION OF
THE SUBSTRING:  9
THE SUBSTRING IS:  MAGIC CAR
```

STR$LEN_EXTR—Extract a Substring of a String

STR$LEN_EXTR copies a substring of a source string into a destination string.

FORMAT

STR$LEN_EXTR *dst-str ,src-str ,start-pos ,length*

corresponding jsb entry point

STR$LEN_EXTR_R8

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

dst-str

VMS Usage:	**char_string**
type:	**character string**
access:	**write only**
mechanism:	**by descriptor**

Destination string into which STR$LEN_EXTR copies the substring. The **dst-str** argument is the address of a descriptor pointing to the destination string.

src-str

VMS Usage:	**char_string**
type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

Source string from which STR$LEN_EXTR extracts the substring that it copies into the destination string. The **src-str** argument is the address of a descriptor pointing to the source string.

start-pos

VMS Usage:	**longword_signed**
type:	**longword integer (signed)**
access:	**read only**
mechanism:	**by reference**

Relative position in the source string at which the substring that STR$LEN_EXTR copies starts. The **start-pos** argument is the address of a signed longword integer containing the starting position.

length

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Number of characters in the substring that STR$LEN_EXTR copies to the destination string. The **length** argument is the address of a signed longword integer containing the length of the substring.

DESCRIPTION

STR$LEN_EXTR extracts a substring from a source string and copies that substring into a destination string.

STR$LEN_EXTR defines the substring by specifying the relative starting position in the source string and the number of characters to be copied. The source string is unchanged, unless it is also the destination string.

If the starting position is less than 1, 1 is used. If the starting position is greater than the length of the source string, the null string is returned. If the length is less than 1, the null string is also returned.

Other routines that may be used to extract and copy a substring are STR$RIGHT, STR$LEFT and STR$POS_EXTR.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
STR$_ILLSTRPOS	STR$LEN_EXTR completed successfully, except that an argument referenced a character position outside the specified string. A default value was used.
STR$_ILLSTRSPE	STR$LEN_EXTR completed successfully, except that the length was too long for the specified string. Default values were used.
STR$_NEGSTRLEN	STR$LEN_EXTR completed successfully, except that **length** contained a negative value. Zero was used.
STR$_TRU	String truncation warning. The fixed-length destination string could not contain all the characters copied from the source string.

CONDITION VALUES SIGNALED

STR$_FATINTERR	Fatal internal error. An internal consistency check has failed. This usually indicates an internal error in the Run-Time Library and should be reported to DIGITAL in a Software Performance Report (SPR).
STR$_ILLSTRCLA	Illegal string class. The class code found in the class field of a descriptor is not a string class code allowed by the VAX Procedure Calling and Condition Handling Standard.
STR$_INSVIRMEM	Insufficient virtual memory. STR$LEN_EXTR could not allocate heap storage for a dynamic or temporary string.

EXAMPLE

```
        CHARACTER*131    IN_STRING
        CHARACTER*1      FRONT_CHAR
        CHARACTER*1      TAIL_CHAR
        INTEGER STR$LEN_EXTR, STR$REPLACE, STR$TRIM
        INTEGER FRONT_POSITION, TAIL_POSITION
10      WRITE (6, 800)
800     FORMAT (' Enter a string, 131 characters or less:',$)
        READ (5, 900, END=200) IN_STRING
900     FORMAT (A)
        ISTATUS = STR$TRIM (IN_STRING, IN_STRING, LENGTH)
        DO 100 I = 1, LENGTH/2
        FRONT_POSITION = I
        TAIL_POSITION = LENGTH + 1 - I
        ISTATUS = STR$LEN_EXTR ( FRONT_CHAR, IN_STRING, FRONT_POSITION,
     A                          %REF(1))
        ISTATUS = STR$LEN_EXTR ( TAIL_CHAR, IN_STRING, TAIL_POSITION,
     A                          %REF(1))
        ISTATUS = STR$REPLACE ( IN_STRING, IN_STRING, FRONT_POSITION,
     A                         FRONT_POSITION, TAIL_CHAR)
        ISTATUS = STR$REPLACE ( IN_STRING, IN_STRING, TAIL_POSITION,
     A                         TAIL_POSITION, FRONT_CHAR)
100     CONTINUE
        WRITE (6, 901) IN_STRING
901     FORMAT (' Reversed string is : ',/,1X,A)
        GOTO 10
200     CONTINUE
        END
```

This FORTRAN program accepts a string as input and writes the string in reverse order as output. This program continues to prompt for input until CTRL/Z is pressed. One sample of the output generated by this program is as follows:

```
$ FORTRAN REVERSE
$ LINK REVERSE
$ RUN REVERSE
Enter a string, 131 characters or less:  African elephants often have
flat feet.
 Reversed string is :
 .teef talf evah netfo stnahpele nacirfA
Enter a string, 131 characters or less: CTRL/Z
$
```

STR$MATCH_WILD—Match Wildcard Specification

STR$MATCH_WILD is used to compare a pattern string that includes wildcard characters with a candidate string. It returns a condition value of STR$_MATCH if the strings match and STR$_NOMATCH if they do not match.

FORMAT

STR$MATCH_WILD *cand-str ,pattern-str*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

cand-str

VMS Usage:	**char_string**
type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

String to which the pattern string is compared. The **cand-str** argument is the address of a descriptor pointing to the candidate string.

pattern-str

VMS Usage:	**char_string**
type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

String containing wildcard characters. The **pattern-str** argument is the address of a descriptor pointing to the pattern string. The wildcards in the pattern string are translated when STR$MATCH_WILD searches the candidate string to determine if it matches the pattern string.

DESCRIPTION

STR$MATCH_WILD translates wildcard characters and searches the candidate string to determine if it matches the pattern string. The pattern string may contain either one or both of the two wildcard characters, asterisk (*) and percent (%). The asterisk character is mapped to one or more characters. The percent character is mapped to only one character.

The two wildcard characters that may be used in the pattern string may be used only as wildcards. If the candidate string contains an asterisk or percent character, the condition STR$_NOMATCH is returned, because the wildcard characters are never translated literally.

CONDITION VALUES RETURNED

STR$_MATCH	The candidate string and the pattern string match.
STR$_NOMATCH	The candidate string and the pattern string do not match.

CONDITION VALUE SIGNALED

STR$_ILLSTRCLA	Illegal string class. Severe error. The descriptor of **cand-str** and/or **pattern-str** contains a class code that is not supported by the VAX Procedure Calling and Condition Handling Standard.

EXAMPLE

```
/*
 *  Example program using STR$MATCH_WILD.  (VAX PL/I V2.3)
 *
 *  The following program reads in a master pattern string and then
 *  compares that to input strings until it reaches the end of the
 *  input file.  For each string comparison done, it will print
 *  either 'Matches pattern string' or 'Doesn't match pattern string'.
 */
declare str$match_wild
    external entry (character(*) varying, character(*) varying)
    returns (bit(1));
example: procedure options(main);
    dcl pattern_string character(80) varying;
    dcl test_string character(80) varying;
    on endfile(sysin) stop;
    put skip;
    get list(pattern_string) options(prompt('Pattern string> '));
    do while( '1'b );
        get skip list(test_string) options(prompt('Test string> '));
        if str$match_wild(test_string,pattern_string)
            then put skip list('Matches pattern string');
            else put skip list('Doesn''t match pattern string');
        end;
    end;
```

This PL/I program demonstrates the use of STR$MATCH_WILD. The output generated by this program is as follows:

```
$ PLI MATCH
$ LINK MATCH
$ RUN MATCH
Pattern string>  'Must match me exactly.'
Test string>  'Will this work? Must match me exactly.'
Doesn't match pattern string
Test string> 'must match me exactly'
Doesn't match pattern string
Test string>  'must match me exactly.'
Doesn't match pattern string
Test string>  'Must match me exactly'
Doesn't match pattern string
Test String>  'Must match me exactly.'
Matches pattern string
```

STR$MUL—Multiply Two Decimal Strings

STR$MUL multiplies two decimal strings.

FORMAT

STR$MUL *asign ,aexp ,adigits ,bsign ,bexp ,bdigits ,csign ,cexp ,cdigits*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

asign

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Sign of the first operand. The **asign** argument is the address of an unsigned longword containing the first operand's sign. Zero is considered positive; 1 is considered negative.

aexp

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Power of 10 by which **adigits** has to be multiplied to get the absolute value of the first operand. The **aexp** argument is the address of a signed longword integer containing this exponent.

adigits

VMS Usage: **char_string**
type: **num. string, unsigned**
access: **read only**
mechanism: **by descriptor**

First operand's numeric string. The **adigits** argument is the address of a descriptor pointing to the numeric string of the first operand. The string must be an unsigned decimal number.

bsign

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Sign of the second operand. The **bsign** argument is the address of an unsigned longword containing the sign of the second operand. Zero is considered positive; 1 is considered negative.

bexp
VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Power of 10 by which **bdigits** has to be multiplied to get the absolute value of the second operand. The **bexp** argument is the address of a signed longword integer containing this exponent.

bdigits
VMS Usage: **char_string**
type: **num. string, unsigned**
access: **read only**
mechanism: **by descriptor**

Second operand's numeric string. The **bdigits** argument is the address of a descriptor pointing to the second operand's numeric string. The string must be an unsigned decimal number.

csign
VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **write only**
mechanism: **by reference**

Sign of the result. The **csign** argument is the address of a signed longword integer containing the sign of the result. Zero is considered positive; 1 is considered negative.

cexp
VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **write only**
mechanism: **by reference**

Power of 10 by which **cdigits** has to be multiplied to get the absolute value of the result. The **cexp** argument is the address of a signed longword integer containing this exponent.

cdigits
VMS Usage: **char_string**
type: **num. string, unsigned**
access: **write only**
mechanism: **by descriptor**

Result's numeric string. The **cdigits** argument is the address of a descriptor pointing to the numeric string of the result. The string will be an unsigned decimal number.

DESCRIPTION
STR$MUL multiplies two decimal strings. The numbers to be multiplied are passed to STR$MUL in three parts: (1) the numeric string, (2) the power of 10 needed to obtain the absolute value, and (3) the sign of the decimal number. The result of the multiplication is also returned in those three parts.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
STR$_TRU	String truncation warning. The fixed-length destination string could not contain all the characters.

CONDITION VALUES SIGNALED

LIB$_INVARG	Invalid argument.
STR$_FATINTERR	Fatal internal error. An internal consistency check has failed. This usually indicates an internal error in the Run-Time Library and should be reported to DIGITAL in a Software Performance Report (SPR).
STR$_ILLSTRCLA	Illegal string class. The class code found in the class field of a descriptor is not a string class code allowed by the VAX Procedure Calling and Condition Handling Standard.
STR$_INSVIRMEM	Insufficient virtual memory. STR$MUL could not allocate heap storage for a dynamic or temporary string.
STR$_WRONUMARG	Wrong number of arguments.

EXAMPLE

```
100 !+
    !  This example program uses
    !  STR$MUL to multiply two decimal
    !  strings (A and B) and place the
    !  results in a third decimal string,
    !  (C)
    !-
    ASIGN% = 1%
    AEXP% = 3%
    ADIGITS$ = '1'
    BSIGN% = 0%
    BEXP% = -4%
    BDIGITS$ = '2'
    CSIGN% = 0%
    CEXP% = 0%
    CDIGITS$ = '0'
    PRINT "A = "; ASIGN%; AEXP%; ADIGITS$
    PRINT "B = "; BSIGN%; BEXP%; BDIGITS$
    CALL STR$MUL        (ASIGN%, AEXP%, ADIGITS$, &
                         BSIGN%, BEXP%, BDIGITS$,  &
                         CSIGN%, CEXP%, CDIGITS$)
    PRINT "C = "; CSIGN%; CEXP%; CDIGITS$
999 END
```

This BASIC example uses STR$MUL to multiply two decimal strings, where the following values apply:

A = –1000 (ASIGN = 1, AEXP = 3, ADIGITS = '1')
B = .0002 (BSIGN = 0, BEXP = –4, BDIGITS = '2')

Listed below is the output generated by this program; note that the decimal value C equals -.2 (CSIGN = 1, CEXP = −1, CDIGITS = 2).

```
A = 1  3 1
B = 0 -4 2
C = 1 -1 2
```

STR$POSITION—Return Relative Position of Substring

STR$POSITION searches for the first occurrence of a single substring within a source string. If STR$POSITION finds the substring, it returns the relative position of that substring. If the substring is not found, STR$POSITION returns a zero.

FORMAT

STR$POSITION *src-str ,sub-str [,start-pos]*

corresponding jsb entry point **STR$POSITION_R6**

RETURNS

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Relative position of the first character of the substring. Zero is the value returned if STR$POSITION did not find the substring.

ARGUMENTS

src-str

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**
Source string within which STR$POSITION searches for the substring. The **src-str** argument is the address of a descriptor pointing to the source string.

sub-str

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**
Substring for which STR$POSITION searches. The **sub-str** argument is the address of a descriptor pointing to the substring.

start-pos

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**
Relative position in the source string at which STR$POSITION begins the search. The **start-pos** argument is the address of a signed longword integer containing the starting position. Although this is an optional argument, it is required if you are using the JSB entry point.

If **start-pos** is not supplied, STR$POSITION starts the search at the first character position of **src-str**.

DESCRIPTION

STR$POSITION returns the relative position of the first occurrence of a substring in the source string. The value returned is an unsigned integer longword. The relative character positions are numbered 1, 2, 3, and so on. Zero indicates that the substring was not found.

If the substring has a zero length, the minimum value of **start-pos** and (the length of **src-str** plus one) is returned by STR$POSITION.

If the source string has a zero length and the substring has a nonzero length, zero is returned, indicating that the substring was not found.

CONDITION VALUE SIGNALED

STR$_ILLSTRCLA	Illegal string class. The class code found in the string class field of a descriptor is not a string class code allowed by the Vax Procedure Calling and Condition Handling Standard.

EXAMPLE

```
PROGRAM POSITION(INPUT,OUTPUT);
{+}
{  This example uses STR$POSITION to determine
{  the position of a the first occurence of
{  a substring (SUBSTRING) within a source
{  string (STRING1) after the starting
{  position (START).
{
{  First, declare the external function.
{-}
FUNCTION STR$POSITION(SRCSTR : VARYING [A]
         OF CHAR; SUBSTR : VARYING [B] OF CHAR;
         STARTPOS : INTEGER) : INTEGER; EXTERN;
{+}
{  Declare the variables used in the main program.
{-}
VAR
  STRING1        : VARYING [256] OF CHAR;
  SUBSTRING      : VARYING [256] OF CHAR;
  START          : INTEGER;
  RET_STATUS     : INTEGER;
{+}
{  Begin the main program. Read the string and substring.
{  Set START equal to 1 to begin looking for the substring
{  at the beginning of the source string.  Call STR$POSITION
{  and print the result.
{-}
BEGIN
  WRITELN('ENTER THE STRING: ');
  READLN(STRING1);
  WRITELN('ENTER THE SUBSTRING: ');
  READLN(SUBSTRING);
  START := 1;
  RET_STATUS := STR$POSITION(STRING1, SUBSTRING, START);
  WRITELN(RET_STATUS);
END.
```

This PASCAL program demonstrates the use of STR$POSITION. If you run this program and set STRING1 equal to KITTEN and substring equal to TEN, the value of RET_STATUS will be 4.

STR$POS_EXTR—Extract a Substring of a String

STR$POS_EXTR copies a substring of a source string into a destination string.

FORMAT

STR$POS_EXTR *dst-str ,src-str ,start-pos ,end-pos*

corresponding jsb entry point

STR$POS_EXTR_R8

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

dst-str

VMS Usage:	**char_string**
type:	**character string**
access:	**write only**
mechanism:	**by descriptor**

Destination string into which STR$POS_EXTR copies the substring. The **dst-str** argument is the address of a descriptor pointing to the destination string.

src-str

VMS Usage:	**char_string**
type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

Source string from which STR$POS_EXTR extracts the substring that it copies into the destination string. The **src-str** argument is the address of a descriptor pointing to the source string.

start-pos

VMS Usage:	**longword_signed**
type:	**longword integer (signed)**
access:	**read only**
mechanism:	**by reference for CALL entry point, by value for JSB entry point**

Relative position in the source string at which the substring that STR$POS_EXTR copies starts. The **start-pos** argument is the address of a signed longword integer containing the starting position.

end-pos

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference for CALL entry point, by value for JSB entry point**

Relative position in the source string at which the substring that STR$POS_EXTR copies ends. The **end-pos** argument is the address of a signed longword integer containing the ending position.

DESCRIPTION

STR$POS_EXTR extracts a substring from a source string and copies the substring into a destination string. STR$POS_EXTR defines the substring by specifying the relative starting and ending positions in the source string. The source string is unchanged, unless it is also the destination string.

If the starting position is less than 1 then 1 is used. If the starting position is greater than the length of the source string, the null string is returned. If the ending position is greater than the length of the source string, the length of the source string is used.

Other routines that may be used to copy a substring are STR$LEFT, STR$RIGHT and STR$LEN_EXTR.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
STR$_ILLSTRPOS	Routine successfully completed, except that an argument referenced a character position outside the specified string. A default value was used.
STR$_ILLSTRSPE	Routine successfully completed, except that **end-pos** was less than **start-pos**. Default values were used.
STR$_TRU	String truncation warning. The fixed-length destination string could not contain all the characters copied from the source string.

CONDITION VALUES SIGNALED

STR$_FATINTERR	Fatal internal error. An internal consistency check has failed. This usually indicates an internal error in the Run-Time Library and should be reported to DIGITAL in a Software Performance Report (SPR).
STR$_ILLSTRCLA	Illegal string class. The class code found in the class field of a descriptor is not a string class code allowed by the VAX Procedure Calling and Condition Handling Standard.
STR$_INSVIRMEM	Insufficient virtual memory. STR$POS_EXTR could not allocate heap storage for a dynamic or temporary string.

EXAMPLE

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
     FTTY     D   F      80            TTY
     C* Initialize source string and position
     C                     MOVE '7 SW Ave'SOURCE  8
     C                     Z-ADD3         BEGPOS  90
     C                     Z-ADD4         ENDPOS  90
     C           POS_EXTR  EXTRN'STR$POS_EXTR'
     C* Extract the 2 character string beginning at position 3
     C                     CALL POS_EXTR
     C                     PARMD          DEST    2
     C                     PARMD          SOURCE
     C                     PARM           BEGPOS     RL
     C                     PARM           ENDPOS     RL
     C* Display on the terminal the extracted string
     C           DEST      DSPLYTTY
     C                     SETON                     LR
```

The RPG II program above displays the string 'SW' on the terminal.

STR$PREFIX—Prefix a String

STR$PREFIX inserts a source string at the beginning of a destination string. The destination string must be dynamic or varying.

FORMAT

STR$PREFIX *dst-str ,src-str*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

dst-str

VMS Usage:	**char_string**
type:	**character string**
access:	**write only**
mechanism:	**by descriptor**

Destination string (dynamic or varying); STR$PREFIX copies the source string into the beginning of this destination string. The **dst-str** argument is the address of a descriptor pointing to the destination string.

src-str

VMS Usage:	**char_string**
type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

Source string which STR$PREFIX copies into the beginning of the destination string. The **src-str** argument is the address of a descriptor pointing to the source string.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed
STR$_TRU	String truncation warning. The fixed-length destination string could not contain all of the characters.

CONDITION VALUES SIGNALED

STR$_FATINTERR	Fatal internal error. An internal consistency check has failed. This usually indicates an internal error in the Run-Time Library and should be reported to DIGITAL in a Software Performance Report (SPR).
STR$_ILLSTRCLA	Illegal string class. The class code found in the class field of a descriptor is not a string class code allowed by the VAX Procedure Calling and Condition Handling Standard.
STR$_INSVIRMEM	Insufficient virtual memory. STR$PREFIX could not allocate heap storage for a dynamic or temporary string.

EXAMPLE

```
10 !+
   ! This example uses STR$PREFIX to
   ! prefix a destination string (D$)
   ! with a source string ('ABCD').
   !-
   EXTERNAL INTEGER FUNCTION STR$PREFIX
   D$ = 'EFG'
   STATUS% = STR$PREFIX (D$, 'ABCD')
   PRINT D$
   END
```

These BASIC statements set D$ equal to 'ABCDEFG'.

STR$RECIP—Reciprocal of a Decimal String

STR$RECIP takes the reciprocal of the first decimal string to the precision limit specified by the second decimal string and returns the result as a decimal string.

FORMAT

STR$RECIP *asign ,aexp ,adigits ,bsign ,bexp ,bdigits ,csign ,cexp ,cdigits*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

asign

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Sign of the first operand. The **asign** argument is the address of an unsigned longword containing the first operand's sign. Zero is considered positive; 1 is considered negative.

aexp

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Power of 10 by which **adigits** has to be multiplied to get the absolute value of the first operand. The **aexp** argument is the address of a signed longword integer containing this exponent.

adigits

VMS Usage: **char_string**
type: **num. string, unsigned**
access: **read only**
mechanism: **by descriptor**

First operand's numeric string. The **adigits** argument is the address of a descriptor pointing to the first operand's numeric string. The string must be an unsigned decimal number.

bsign
VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**
Sign of the second operand. The **bsign** argument is the address of an unsigned longword containing the sign of the second operand. Zero is considered positive; 1 is considered negative.

bexp
VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**
Power of 10 by which **bdigits** has to be multiplied to get the absolute value of the second operand. The **bexp** argument is the address of a signed longword integer containing this exponent.

bdigits
VMS Usage: **char_string**
type: **num. string, unsigned**
access: **read only**
mechanism: **by descriptor**
Second operand's numeric string. The **bdigits** argument is the address of a descriptor pointing to the second operand's numeric string. The string must be an unsigned decimal number.

csign
VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **write only**
mechanism: **by reference**
Sign of the result. The **csign** argument is the address of a signed longword integer containing the result's sign. Zero is considered positive; 1 is considered negative.

cexp
VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **write only**
mechanism: **by reference**
Power of 10 by which **cdigits** has to be multiplied to get the absolute value of the result. The **cexp** argument is the address of a signed longword integer containing this exponent.

cdigits

VMS Usage: **char_string**
type: **num. string, unsigned**
access: **write only**
mechanism: **by descriptor**

Result's numeric string. The **cdigits** argument is the address of a descriptor pointing to the result's numeric string. The string will be an unsigned decimal number.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
STR$_TRU	String truncation warning. The fixed-length destination string could not contain all of the characters.

CONDITION VALUES SIGNALED

STR$_DIVBY_ZER	Division by zero.
LIB$_INVARG	Invalid argument.
STR$_FATINTERR	Fatal internal error. An internal consistency check has failed. This usually indicates an internal error in the Run-Time Library and should be reported to DIGITAL in a Software Performance Report (SPR).
STR$_ILLSTRCLA	Illegal string class. The class code found in the class field of a descriptor is not a string class code allowed by the VAX Procedure Calling and Condition Handling Standard.
STR$_INSVIRMEM	Insufficient virtual memory. STR$RECIP could not allocate heap storage for a dynamic or temporary string.
STR$_WRONUMARG	Wrong number of arguments.

EXAMPLE

```
100 !+
    ! This example program uses
    ! STR$RECIP to find the reciprocal of
    ! the first decimal string (A) to the
    ! precision specified in the second
    ! decimal string (B), and place the
    ! result in a third decimal string (C).
    !-

    ASIGN% = 1%
    AEXP% = 3%
    ADIGITS$ = '1'
    BSIGN% = 0%
    BEXP% = -4%
    BDIGITS$ = '2'
    CSIGN% = 0%
    CEXP% = 0%
    CDIGITS$ = '0'
```

```
    PRINT "A = "; ASIGN%; AEXP%; ADIGITS$
    PRINT "B = "; BSIGN%; BEXP%; BDIGITS$
    CALL STR$RECIP      (ASIGN%, AEXP%, ADIGITS$, &
                         BSIGN%, BEXP%, BDIGITS$, &
                         CSIGN%, CEXP%, CDIGITS$)
    PRINT "C = "; CSIGN%; CEXP%; CDIGITS$
999 END
```

This BASIC example uses STR$RECIP to find the reciprocal of A to the precision level specified in B.

The following values apply:

A = –1000 (ASIGN = 1, AEXP = 3, ADIGITS = '1')
B = .0002 (BSIGN = 0, BEXP = –4, BDIGITS = '2')

The output generated by this program is as follows, yielding a decimal value of C equal to –.001.

```
A = 1  3 1
B = 0 -4 2
C = 1 -3 1
```

STR$REPLACE—Replace a Substring

STR$REPLACE copies a source string to a destination string, replacing part of the string with another string. The substring to be replaced is specified by its starting and ending positions.

FORMAT

STR$REPLACE *dst-str ,src-str ,start-pos ,end-pos ,rpl-str*

corresponding jsb entry point **STR$REPLACE_R8**

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

dst-str

VMS Usage:	**char_string**
type:	**character string**
access:	**write only**
mechanism:	**by descriptor**

Destination string into which STR$REPLACE writes the new string created when it replaces the substring. The **dst-str** argument is the address of a descriptor pointing to the destination string.

src-str

VMS Usage:	**char_string**
type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

Source string. The **src-str** argument is the address of a descriptor pointing to the source string.

start-pos

VMS Usage:	**longword_signed**
type:	**longword integer (signed)**
access:	**read only**
mechanism:	**by reference for CALL entry point, by value for JSB entry point**

Position in the source string at which the substring which STR$REPLACE replaces begins. The **start-pos** argument is the address of a signed longword integer containing the starting position. The position is relative to the start of the source string.

end-pos

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference for CALL entry point, by value for JSB entry point**

Position in the source string at which the substring which STR$REPLACE replaces ends. The **end-pos** argument is the address of a signed longword integer containing the ending position. The position is relative to the start of the source string.

rpl-str

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Replacement string with which STR$REPLACE replaces the substring. The **rpl-str** argument is the address of a descriptor pointing to this replacement string.

DESCRIPTION

STR$REPLACE copies a source string to a destination string, replacing part of the string with another string. The substring to be replaced is specified by its starting and ending positions.

If the starting position is less than 1, 1 is used. If the ending position is greater than the length of the source string, the length of the source string is used. If the starting position is greater than the ending position, the overlapping portion of the source string will be copied twice.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
STR$_ILLSTRPOS	Routine successfully completed, but an argument referenced a character position outside the specified string. A default value was used.
STR$_ILLSTRSPE	Routine successfully completed, but **end-pos** was less than **start-pos** or length was too long for the specified string. Default values were used.
STR$_TRU	String truncation warning. The fixed-length destination string could not contain all of the characters.

CONDITION VALUES SIGNALED

STR$_FATINTERR	Fatal internal error. An internal consistency check has failed. This usually indicates an internal error in the Run-Time Library and should be reported to DIGITAL in a Software Performance Report (SPR).
STR$_ILLSTRCLA	Illegal string class. The class code found in the class field of a descriptor is not a string class code allowed by the VAX Procedure Calling and Condition Handling Standard.
STR$_INSVIRMEM	Insufficient virtual memory. STR$REPLACE could not allocate heap storage for a dynamic or temporary string.

EXAMPLE

```
10 !+
   ! This examples uses STR$REPLACE to
   ! replace all characters from the starting
   ! position (2%) to the ending position (3%)
   ! with characters from the replacement string
   ! ('XYZ').
   !-
   EXTERNAL INTEGER FUNCTION STR$REPLACE
   D$ = 'ABCD'
   STATUS% = STR$REPLACE (D$, D$, 2%, 3%, 'XYZ')
   PRINT D$
   END
```

These BASIC statements set D$ equal to 'AXYZD'.

STR$RIGHT—Extract a Substring of a String

STR$RIGHT copies a substring of a source string into a destination string.

FORMAT

STR$RIGHT *dst-str ,src-str ,start-pos*

corresponding jsb entry point **STR$RIGHT_R8**

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

dst-str

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

Destination string into which STR$RIGHT copies the substring. The **dst-str** argument is the address of a descriptor pointing to the destination string.

src-str

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Source string from which STR$RIGHT extracts the substring that it copies into the destination string. The **src-str** argument is the address of a descriptor pointing to the source string.

start-pos

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference for CALL entry point, by value for JSB entry point**

Relative position in the source string at which the substring that STR$RIGHT copies starts. The **start-pos** argument is the address of a signed longword integer containing the starting position.

DESCRIPTION

STR$RIGHT extracts a substring from a source string and copies that substring into a destination string. STR$RIGHT defines the substring by specifying the relative starting position. The relative ending position is equal to the length of the source string. The source string is unchanged, unless it is also the destination string.

If the starting position is less than 2, the entire source string is copied. If the starting position is greater than the length of the source string, a null string is copied.

This is a variation of STR$POS_EXTR. Other routines that may be used to extract and copy a substring are STR$LEFT and STR$LEN_EXTR.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
STR$_ILLSTRPOS	Routine successfully completed, except that an argument referenced a character position outside the specified string. A default value was used.
STR$_TRU	String truncation warning. The fixed-length destination string could not contain all the characters copied from the source string.

CONDITION VALUES SIGNALED

STR$_FATINTERR	Fatal internal error. An internal consistency check has failed. This usually indicates an internal error in the Run-Time Library and should be reported to DIGITAL in a Software Performance Report (SPR).
STR$_ILLSTRCLA	Illegal string class. The class code found in the class field of a descriptor is not a string class code allowed by the VAX Procedure Calling and Condition Handling Standard.
STR$_INSVIRMEM	Insufficient virtual memory. STR$RIGHT could not allocate heap storage for a dynamic or temporary string.

EXAMPLE

```
PROGRAM RIGHT(INPUT, OUTPUT);
{+}
{  This example uses STR$RIGHT to extract a substring
{  from a specified starting position (START_POS) to
{  the end (right side) of a source string (SRC_STR)
{  and write the result in a destination string (DST_STR).
{
{  First, declare the external procedure.
{-}
PROCEDURE STR$RIGHT(%DESCR DSTSTR: VARYING
          [A] OF CHAR; SRCSTR  : VARYING [B] OF CHAR;
          STARTPOS : INTEGER); EXTERN;
{+}
{  Declare the variables used in the main program.
{-}
VAR
  SRC_STR       : VARYING [256] OF CHAR;
  DST_STR       : VARYING [256] OF CHAR;
  START_POS     : INTEGER;
```

```
{+}
{  Begin the main program.  Read the source string
{  and starting position. Call STR$RIGHT to extract
{  the substring.  Print the result.
{-}
BEGIN
  WRITELN('ENTER THE SOURCE STRING: ');
  READLN(SRC_STR);
  WRITELN('ENTER THE STARTING POSITION');
  WRITELN('OF THE SUBSTRING: ');
  READLN(START_POS);
  STR$RIGHT(DST_STR, SRC_STR, START_POS);
  WRITELN;
  WRITELN('THE SUBSTRING IS: ',DST_STR);
END.
```

This PASCAL program uses STR$RIGHT to extract a substring from a specified starting position (START_POS) to the end of the source string. One sample of the output is as follows:

```
$ RUN RIGHT
ENTER THE SOURCE STRING: BLUE PLANETS ALWAYS HAVE PURPLE PLANTS
ENTER THE STARTING POSITION
OF THE SUBSTRING: 27
THE SUBSTRING IS: URPLE PLANTS
```

STR$ROUND—Round or Truncate a Decimal String

STR$ROUND rounds or truncates a decimal string to a specified number of significant digits and places the result in another decimal string.

FORMAT

STR$ROUND *places ,trunc-flg ,asign ,aexp ,adigits ,csign ,cexp ,cdigits*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

places

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Maximum number of decimal digits that STR$ROUND retains in the result. The **places** argument is the address of a signed longword integer containing the number of decimal digits.

trunc-flg

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Function flag. Zero indicates that the decimal string is rounded; 1 indicates that it is truncated. The **trunc-flg** argument is the address of an unsigned longword containing this function flag.

asign

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

Sign of the first operand. The **asign** argument is the address of an unsigned longword string containing this sign. A value of zero indicates that the number is positive, while a value of 1 indicates that the number is negative.

aexp

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Power of 10 by which **adigits** has to be multiplied to get the absolute value of the first operand. The **aexp** argument is the address of a signed longword integer containing this exponent.

adigits

VMS Usage: **char_string**
type: **num. string, unsigned**
access: **read only**
mechanism: **by descriptor**

First operand's numeric string. The **adigits** argument is the address of a descriptor pointing to this numeric string. The string must be an unsigned decimal number.

csign

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **write only**
mechanism: **by reference**

Sign of the result. The **csign** argument is the address of a signed longword integer containing the result's sign. A value of zero indicates that the number is positive, while a value of 1 indicates that the number is negative.

cexp

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **write only**
mechanism: **by reference**

Power of 10 by which **cdigits** has to be multiplied to get the absolute value of the result. The **cexp** argument is the address of a signed longword integer containing this exponent.

cdigits

VMS Usage: **char_string**
type: **num. string, unsigned**
access: **write only**
mechanism: **by descriptor**

Result's numeric string. The **cdigits** argument is the address of a descriptor pointing to this numeric string. The string will be an unsigned decimal number.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
STR$_TRU	String truncation warning. The fixed-length destination string could not contain all of the characters.

CONDITION VALUES SIGNALED

LIB$_INVARG	Invalid argument.
STR$_FATINTERR	Fatal internal error. An internal consistency check has failed. This usually indicates an internal error in the Run-Time Library and should be reported to DIGITAL in a Software Performance Report (SPR).
STR$_ILLSTRCLA	Illegal string class. The class code found in the class field of a descriptor is not a string class code allowed by the VAX Procedure Calling and Condition Handling Standard.
STR$_INSVIRMEM	Insufficient virtual memory. STR$ROUND could not allocate heap storage for a dynamic or temporary string.
STR$_WRONUMARG	Wrong number of arguments.

EXAMPLE

```
100 !+
    ! This example shows the difference between
    ! the values obtained when rounding or truncating
    ! a decimal string.
    !-
    ASIGN% = 0%
    AEXP% = -4%
    ADIGITS$ = '9999998'
    CSIGN% = 0%
    CEXP% = 0%
    CDIGITS$ = '0'
    PRINT "A = "; ASIGN%; AEXP%; ADIGITS$
    !+
    ! First, call STR$ROUND to round the value of A.
    !-
    CALL STR$ROUND      (3%, 0%, ASIGN%, AEXP%, ADIGITS$, &
                        CSIGN%, CEXP%, CDIGITS$)
    PRINT "ROUNDED:  C = "; CSIGN%; CEXP%; CDIGITS$
    !+
    ! Now, call STR$ROUND to truncate the value of A.
    !-
    CALL STR$ROUND      (3%, 1%, ASIGN%, AEXP%, ADIGITS$, &
                        CSIGN%, CEXP%, CDIGITS$)
    PRINT "TRUNCATED:  C = "; CSIGN%; CEXP%; CDIGITS$
999 END
```

This BASIC example uses STR$ROUND to first round and then truncate the value of A to the number of decimal places specified by **places**. The following values apply:

A = 999.9998 (ASIGN = 1, AEXP = –4, ADIGITS = '9999998')

Listed below is the output generated by this program; note that the decimal value of C equals 1000 when rounded, and 999 when truncated.

```
A = 1  -4 9999998
ROUNDED: C = 0 1 100
TRUNCATED: C = 0 0 999
```

STR$TRANSLATE—Translate Matched Characters

STR$TRANSLATE successively compares each character in a source string to all characters in a match string. If a source character has a match, the destination character is taken from the translate string. Otherwise, STR$TRANSLATE moves the source character to the destination string.

FORMAT

STR$TRANSLATE *dst-str ,src-str ,trans-str ,match-str*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

dst-str

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

Destination string. The **dst-str** argument is the address of a descriptor pointing to the destination string.

src-str

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Source string. The **src-str** argument is the address of a descriptor pointing to the source string.

trans-str

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Translate string. The **trans-str** argument is the address of a descriptor pointing to the translate string.

match-str

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Match string. The **match-str** argument is the address of a descriptor pointing to the match string.

DESCRIPTION

STR$TRANSLATE successively compares each character in a source string to all characters in a match string. If a source character matches any of the characters in the match string, STR$TRANSLATE moves a character from the translate string to the destination string. Otherwise, STR$TRANSLATE moves the character from the source string to the destination string.

The character taken from the translate string has the same relative position as the matching character had in the match string. When a character appears more than once in the match string, the position of the leftmost occurrence of the multiply defined character is used to select the translate string character. If the translate string is shorter than the match string and the matched character position is greater than the translate string length, the destination character is a space.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
STR$_TRU	String truncation warning. The fixed-length destination string could not contain all of the characters.

CONDITION VALUES SIGNALED

STR$_FATINTERR	Fatal internal error. An internal consistency check has failed. This usually indicates an internal error in the Run-Time Library and should be reported to DIGITAL in a Software Performance Report (SPR).
STR$_ILLSTRCLA	Illegal string class. The class code found in the class field of a descriptor is not a string class code allowed by the VAX Procedure Calling and Condition Handling Standard.
STR$_INSVIRMEM	Insufficient virtual memory. STR$TRANSLATE could not allocate heap storage for a dynamic or temporary string.

EXAMPLE

```
10      !+
        !  This example program uses STR$TRANSLATE to
        !  translate all characters of a source string
        !  from uppercase to lowercase characters.
        !-
        EXTERNAL INTEGER FUNCTION STR$TRANSLATE(STRING,STRING,STRING,STRING)
        TO$='abcdefghijklmnopqrstuvwxyz'
        FROM$='ABCDEFGHIJKLMNOPQRSTUVWXYZ'
        X% = STR$TRANSLATE(OUT$,'TEST',TO$,FROM$)
        PRINT 'Status = ';x%
        PRINT 'Resulting string = ';out$
32767   END
```

This BASIC example translates uppercase letters to lowercase letters, thus performing the same function as STR$UPCASE.

The output generated by this example is as follows:

```
$ RUN TRANSLATE
Status =  1
Resulting string = test
```

A more practical although more complicated use for STR$TRANSLATE would be to encrypt data by translating the characters to obscure combinations of numbers and alphabetic characters.

STR$TRIM—Trim Trailing Blanks and Tabs

STR$TRIM copies a source string to a destination string and deletes the trailing blank and tab characters.

FORMAT

STR$TRIM *dst-str ,src-str [,out-len]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

dst-str

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

Destination string into which STR$TRIM copies the trimmed string. The **dst-str** argument is the address of a descriptor pointing to the destination string.

src-str

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Source string which STR$TRIM trims and then copies into the destination string. The **src-str** argument is the address of a descriptor pointing to the source string.

out-len

VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

Number of bytes that STR$TRIM has written into **dst-str**, not counting padding in the case of a fixed-length string. The **out-len** argument is the address of an unsigned word into which STR$TRIM writes the length of the output string. If the input string is truncated to the size specified in the **dst-str** description, **out-len** is set to this size. Therefore, **out-len** can always be used by the calling program to access a valid substring of **dst-str**.

CONDITION VALUES RETURNED	SS$_NORMAL	Routine successfully completed.
	STR$_TRU	String truncation warning. The fixed-length destination string could not contain all the characters.

CONDITION VALUES SIGNALED	STR$_FATINTERR	Fatal internal error. An internal consistency check has failed. This usually indicates an internal error in the Run-Time Library and should be reported to DIGITAL in a Software Performance Report (SPR).
	STR$_ILLSTRCLA	Illegal string class. The class code found in the class field of a descriptor is not a string class code allowed by the VAX Procedure Calling and Condition Handling Standard.
	STR$_INSVIRMEM	Insufficient virtual memory. STR$TRIM could not allocate heap storage for a dynamic or temporary string.

STR$UPCASE—Convert String to All Uppercase Characters

STR$UPCASE converts a source string to uppercase and writes the converted string into the destination string. When you need to compare characters without regard to case, you can first use STR$UPCASE to convert both characters to uppercase. STR$UPCASE converts all characters in the multinational character set.

FORMAT

STR$UPCASE *dst-str ,src-str*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

dst-str

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

Destination string into which STR$UPCASE writes the string it has converted to uppercase. The **dst-str** argument is the address of a descriptor pointing to the destination string.

src-str

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Source string that STR$UPCASE converts to uppercase. The **src-str** argument is the address of a descriptor pointing to the source string.

CONDITION VALUES RETURNED

SS$_NORMAL	Routine successfully completed.
STR$_TRU	String truncation warning. The fixed-length destination string could not contain all the characters.

CONDITION VALUES SIGNALED

STR$_FATINTERR	Fatal internal error. An internal consistency check has failed. This usually indicates an internal error in the Run-Time Library and should be reported to DIGITAL in a Software Performance Report (SPR).
STR$_ILLSTRCLA	Illegal string class. The class code found in the class field of a descriptor is not a string class code allowed by the VAX Procedure Calling and Condition Handling Standard.
STR$_INSVIRMEM	Insufficient virtual memory. STR$UPCASE could not allocate heap storage for a dynamic or temporary string.

EXAMPLES

1

```
30  !+
    !  This example uses STR$UPCASE
    !  to convert all characters in
    !  the source string (SRC$) to
    !  uppercase and write the result
    !  in the destination string (DST$).
    !-

    SRC$ = 'abcd'
    PRINT "SRC$ =";SRC$
    CALL STR$UPCASE (DST$, SRC$)
    PRINT "DST$ =";DST$
    END
```

This BASIC program generates the following output:

```
SCR$ =abcd
DST$ =ABCD
```

2

```
0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890

FTTY     D   F       80             TTY
C* Initialize string to be converted to upper case
C                       MOVE 'rep head'HEAD    8
C             UPCASE    EXTRN'STR$UPCASE'
C* Convert the string to upper case
C                       CALL UPCASE
C                       PARMD           RESULT  8
C                       PARMD           HEAD
C* Display on the terminal the string in upper case
C             RESULT    DSPLYTTY
C                       SETON                      LR
```

The RPG II program above displays the string 'REP HEAD' on the terminal.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

T

U

V

Z

READER'S COMMENTS

Note: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent:

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) ____________

Name ____________ Date ____________

Organization ____________

Street ____________

City ____________ State ____________ Zip Code or Country ____________

Do Not Tear - Fold Here and Tape

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

SSG PUBLICATIONS ZK1-3/J35
DIGITAL EQUIPMENT CORPORATION
110 SPIT BROOK ROAD
NASHUA, NEW HAMPSHIRE 03062-2698

Do Not Tear - Fold Here

Cut Along Dotted Line

VAX/VMS Run-Time Library
Routines Reference Manual
AA-Z502C-TE
AA-Z505C-TE
QLZ55-GZ (Volumes 5C and 5D)

READER'S COMMENTS

Note: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent:

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) ____________

Name ____________ Date ____________

Organization ____________

Street ____________

City ____________ State ____________ Zip Code or Country ____________

— — Do Not Tear - Fold Here and Tape —

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

SSG PUBLICATIONS ZK1-3/J35
DIGITAL EQUIPMENT CORPORATION
110 SPIT BROOK ROAD
NASHUA, NEW HAMPSHIRE 03062-2698

— — Do Not Tear - Fold Here —

Cut Along Dotted Line

NOTES

NOTES